TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PETRI SANDSTRÖM

AIKAAN JA PAIKKAAN SIDOTTU MOBIILIMAKSUSOVELLUS HYÖDYNTÄEN FACEBOOK MESSENGER RAJAPINTAA

Diplomityö

Tarkastaja: professori Kari Systä
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 3. tammi-kuuta 2018

TIIVISTELMÄ

Petri Sandström: Aikaan ja paikkaan sidottu mobiilimaksusovellus hyödyntäen Facebook Messenger rajapintaa
Tampereen teknillinen yliopisto
Diplomityö, 42 sivua
Marraskuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Kari Systä

Avainsanat: Facebook Messenger API, chat bot, node.js, checkout integraatio

Yhä harvemmat ihmiset kantavat mukanaan käteistä. Sen sijaan taskusta löytyy lähes jokaiselta maksukortti tai teknisesti edistyneemmät voivat kantaa mukanaan älypuhelinta maksuvälineenään. Tämän todellisuuden takia seurakunnat ovat etsineet ratkaisua ongelmaan, kun kolehtia on kerätty pääsääntöisesti käteisvaroista.

Kolehdin keruu on rahankeruuluvan alaista toimintaa, jolloin jokainen maksu on pystyttävä kohdentavaan tiettyyn tilaisuuteen sijainnin ja ajan puolesta. Tämän diplomityön tavoitteena on toteuttaa rahankeruuluvan mukainen Facebook Messenger-rajapintaa hyödyntävä mobiilimaksusovellus Tampereen evankelisluterilaisen kirkon jumalanpalveluksiin. Pääpainotus käytännöntyössä, ja tämä dokumentti koostaa työn eri vaiheet, teorian ja käytännön ratkaisut kattavaan raporttiin, jonka lopussa esitetään jatkokehityssuunnitelma.

Valmiin, toimivan mobiilisovelluksen arkkitehtuuri kostuu seuraavista osista: Facebook ryhmä, jonka tokenin avulla yhdistetään Facebook Messenger-rajapinta Chatfuel-palvelun kanssa yhteen. Chatfuel-palvelussa toteutetaan graafisen käyttöliittymän (GUI) avulla keskustelupolku, jota kulkemalla käyttäjältä kysytään tarvittavat tiedot maksunsuorittamista varten. Maksua varten tarvitaan käyttäjän sijainti sekä summa joka halutaan antaa kolehtiin. Näiden tietojen lisäksi maksuikkuna on aukaistu vain tiettyyn aikaan viikosta, jumalanpalvelusten aikana. Keskustelupolun lopussa on linkki, jota painamalla siirrytään Facebook Messenger sovelluksessa verkkopalveluun, jossa varsinainen Checkout-maksujärjestelmä on integroitu maksuväyläksi. Kun maksu on tehty, tai se keskeytyy, palataan verkkopalveluun, jossa kiitetään maksusuorituksesta, tai virheen sattuessa annetaan virheilmoitus.

Työn tutkimusmenetelmänä käytetiin konstruktiivista lähestymistapaa. Tarkoituksena oli löytää määritetyt kriteerit täyttävä kolmannen osapuolen palvelu, jonka avulla voidaan toteuttaa GUI-pohjainen keskustelupolku, joka alittaa teknisen kynnyksen riman tarpeeksi matalalta. Tämän lisäksi yhtenä kriteerinä toimi valmis toteutus sijaintipalveluiden hyödyntämiseen. Checkout-maksujärjestelmän dokumentaatiota hyödynnettiin valitsemalla ohjelmistokirjasto, jota voitiin hyödyntää varsinaisen integraation toteuttamisessa.

ABSTRACT

Petri Sandström: Time and location bound mobile payment software using Facebook Messenger API
Tampere University of Technology
Master of Science Thesis, 42 pages
November 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Kari Systä

Keywords: Facebook Messenger API, chat bot, node.js, checkout integration

People tend to carry less and less cash with them nowadays. Instead, almost everyone carries credit and debit cards for payments, and more tech savvy people even use their smart phones or smart watches to make purchases. Realizing this reality, the congregations have searched for a solution for the problem, as the offerings are collected mainly as cash.

Collecting an offering falls under the fundraising permit law where each payment must be targeted for a certain event by a location and time. The purpose of this Master's thesis is to implement a functional mobile web payment application that works under the fundraising permit. The main focus of the thesis is the software development of the mobile application for payments. This document contains plans for each phase of the software development process, the theory of it, and the practical details. Lastly, we will go through the further development plan.

The architecture consists of the following parts: a Facebook group's token for integrating the group with a third-party service called Chatfuel. The service implements the Facebook Messenger Application Programming Interface (API) as a graphic user interface (GUI) to build the chat flow. In the flow, the user is prompted about his/her location and the amount of the offering. Combining this data with a check of a correct time for a service, the user is forwarded to a website that implements Checkout payment portal integration for secure payments. When the payment flow is finished, the user is forwarded to a provided callback for success or failure message.

Constructive research methodologies were used when planning and implementing the software. Combining this with criteria in regards to ease of use, a GUI-based service that implements a location service handling, quality decisions were made. The Checkout payment portal documentation included a software library that implemented parts of the Checkout API for ease of use when integrating the service.

ALKUSANAT

Tämä diplomityö toteutettiin Tampereen seurakuntayhtymän alaisuudessa toimivalle Uudelle Versolle, jossa mobiilimaksusovellus otettiin käyttöön innokkaasti syksyllä 2017.

Kiitos Olli Viljakaiselle aloitteesta chatbotin toteutuksen suhteen, Ari Viljakaiselle OP:n kontakteihin verkostoitumisesta, ideoinnista botin toiminnallisuuteen ja mittavasta selvityksestä sekä taustatyöstä kirkkolain osalta. Michal Olczakille ensiaskelten teknisten valintojen esittelystä, toimintojen sparrauksesta sekä testauksesta ja palautteesta.

Haluan kiittää tarkastajaa professori Kari Systää hänen kärsivällisyydestä työn aikataulutuksen suhteen sekä avusta diplomityön sisällön jäsentelyyn ja kommentointiin liittyen.

Viimeiseksi haluan kiittää rakasta vaimoani, joka tuki minua sekä kiireellisen ohjelmointiosuuden, että kirjoitusprosessin aikana ja mahdollisti niiden toteutumisen.

Tampereella, 13.11.2018

Petri Sandström

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

API	engl. Application programming interface, rajapinta
TTY	Tampereen teknillinen yliopisto
URL	engl. Uniform Resource Locator, verkkosivun osoite
Checkout	Checkout niminen maksunvälityspalvelu
Keskustelubotti	engl. Chat Bot, ohjelmisto, joka simuloi interaktiivisen keskustelun ihmiskäyttäjän kanssa
CTA	engl. Call-to-action, toimintakehoite-painike
MVP	engl. Minimum Viable Product, prosessi nopealle prototypoinnille, jossa toiminnallisuudet julkaistaan aikaisessa vaiheessa
RSS	engl. Really Simple Syndication, teknologia digitaalisen sisällön julkaisemiseen/jakamiseen
DDoS	engl. Distributed Denial-of-Service, on palvelunestohyökkäys, jossa palvelimelle ohjataan liikaa kutsuja, siten ruuhkauttamalla palvelu
IaaS	Infrastructure as a Service on virtuaalikoneita webin kautta tarjoava palvelu

1. JOHDANTO

Sähköisten maksutapojen ja mobiilimaksamisteknologioiden kehitys aiheuttaa käteisrahan käyttöasteen merkittävän laskun [1]. Tampereella päätettiin ratkaista ongelma käteisrahan käytön hiipumisesta kehittämällä mobiilimaksusovellus. Sovelluksella voidaan kerätä mobiilikolehti Tampereen evankelisluterilaisen kirkon alla toimivan Uuden Verson jumalanpalveluksissa.

Mobiilimaksusovellus on rakennettu noudattamaan nykyistä kirkkolakia ja myönnettyä rahankeruulupaa. Nämä toteutuvat, kun kolehti kerätään tiettyyn tilaisuuteen eli on tiedettävä maksun aika sekä sijainti. Näiden tietojen avulla tapahtuma voidaan yksilöidä tiettyyn kolehdinkeruu tilaisuuteen. Mobiilikolehdin lisäksi tilaisuudessa kiertää myös perinteinen kolehtihaavi, johon voidaan tehdä käteismaksuja. Tämä on määritetty myös kirkkolakiin.

Aiempia kokemuksia maksamisesta ilman käteistä, on muun muassa lähettämällä tekstiviestilahjoitus tai keräämällä maksupäätteellä korttimaksuja. Tekstiviestillä maksaminen olisi kätevää ja helppoa, mutta maksutavan toteutuksen rajaaminen tiettyyn tilaisuuteen ajan ja paikan suhteen on vähintäänkin haastavaa. Tämän lisäksi operaattorit keräävät merkittävän osan lahjoitusten määrästä itselleen. Nämä maksutavat eivät siis ole keränneet suosiota kolehtimaksun suoritukseen, ja tilalle oli löydettävä uusi moderni maksutapa.

Vaatimuksena mobiilisovelluksen käyttöön on, että käyttäjällä on Facebook tili ja Facebook Messenger sovellus asennettuna mobiililaitteellensa. Näiden lisäksi joko OP:n Pivo mobiililompakko tai Danske Bankin MobilePay-sovellus. Vaatimusten täyttymistä seurasi Uuden Verson pastori Viljakainen, joka toimi myös tilaajana tuotteelle prosessin näkökulmasta. Uudet toiminnallisuudet suunniteltiin yhdessä ja varmistettiin lopullinen toteutuminen ja hyväksyntä hänen kauttaan.

Lyhyesti tiivistettynä mobiilikolehdin voi suorittaa seuraavaksi kuvatulla tavalla. Aloita keskustelu Uuden Verson kanssa Facebook Messenger sovelluksella. Keskustelua ohjaa ohjelmoitu keskustelubotti, joka johdattelee käyttäjän askel askeleelta kohti maksusuoritusta. Ensin syötetään laitteen sijaintipalvelun avulla käyttäjän sijainti joka varmennetaan. Jos sijainti täsmää, valitaan haluttu kolehdin summa. Tämän jälkeen keskustelubotti tarjoaa linkin verkkopalveluun, jossa suoritetaan Checkout-maksujärjestelmän avulla varsinainen maksusuoritus. Jos aika ei täsmää olemassa olevaan kolehdinmaksutilaisuuteen, annetaan käyttäjälle virheilmoitus ja seuraavan tilaisuuden ajankohta.

Diplomityönä toimii tämän mobiilimaksusovelluksen toteutus kokonaisuudessaan sekä tämä dokumentti raporttina ja selitteenä työn taustoista. Ensin käymme tarkemmin läpi työn lähtökohdat ja taustatiedot sen tarpeelle sekä perehdytään ratkaistavaan ongelmaan. Kolmannessa luvussa tarkastellaan mitä teknologiavalintoja sovellukseen tehtiin ja arvioidaan olivatko valinnat onnistuneita vaatimuksiin nähden. Kun taustatiedot ja teknologiavalinnat on selvitetty, on vuorossa tarkempi kuvaus varsinaisesta mobiilimaksusovelluksesta. Sen jälkeen käydään läpi maksun suoritus ja keskustelukulku (engl. chat flow). Lopulta esitellään jatkokehityssuunnitelma sovelluksen tuotteistamiseen liittyen ja arvioidaan työn onnistuminen. Viimeiseksi kootaan yhteenveto aiheesta.

2. LÄHTÖKOHDAT JA VAATIMUKSET

Tässä luvussa käydään läpi työn lähtökohdat. Tarkastellaan minkä tarpeen diplomityö täyttää ja miten se on ratkaistu. Esitellään konstruktiivinen metodologia ja määritellään sovellukselle asetettavat vaatimukset. Tarkempi tekninen kuvaus prosessista ja toteutusteknologioista löytyy luvuista kolme ja neljä.

2.1 Taustatietoa

Tämän mobiilimaksusovelluksen ensimmäinen versio on otettu käyttöön syksyllä 2017 Tampereen evankelisluterilaisen kirkon alla toimivan Uuden Verson varikkomessussa eli jumalanpalveluksessa. Se on tiedettävästi ensimmäinen laatuaan valtion kirkon sisällä.

Suomen Pankin rahahuolto-osaston johtava neuvonantaja Kari Takala arvioi [1], että käteinen menettää asemaansa odotettua nopeammin. Syyksi sille, että käteinen syrjäytyy lähimaksamisen tilalta, hän kertoo olevan lähimaksutapahtuman nopeus ja helppous. Tähän ilmiöön Uudella Versolla ollaan herätty, ja ongelma haluttiin ratkaista ennen kuin käteisen vähentyminen näkyy konkreettisesti kolehdin määrässä. Mobiilimaksamista on toivottu seurakuntayhteisön sisällä jo useaan otteeseen, mutta sopivaa keinoa sen toteuttamiseksi ei ole löytynyt aiemmin. Maksukorttipäätteitä on käytetty, mutta maksutapa ei ole tehokkuudeltaan yhtä hyödyllinen kuin mobiilimaksaminen. Maksupäätteitä ei usein ole kovin montaa, ja niiden käyttö on huomattavasti hitaampaa. Mobiilimaksusovellusta käyttämällä käyttäjä voi omalta paikaltaan muutamalla painalluksella suorittaa kolehdin maksun turvallisesti.

Idean keksi Uuden Verson pastori Olli Viljakainen, sillä uusi moderni maksutapa oli saatava. Tämän diplomityön kirjoittaja valittiin projektiin ohjelmistosuunnittelijaksi. Parin viikon suunnittelun pohjalta lähdettiin toteuttamaan ensimmäistä versiota ja hahmottamaan toimintojen teknisiä vaatimuksia. Prosessia lähdettiin viemään eteenpäin Minimum Viable Product (MVP) käytäntöä hyödyntäen. Käytännössä tämä tarkoittaa, että tuotetaan pienellä työllä ja minimaalisin kustannuksin, tässä tapauksessa resurssina aika, uusia versioita sovelluksesta ja testataan uudet toiminnallisuudet. Lisää prosessikokonaisuudesta aliluvussa 4.1.

Varsinaisen ohjelmointityön lisäksi byrokratiaa on hoidettava valitun maksujärjestelmän osalta, sillä valitsimme maksupalveluntarjoajaksi OP:n Checkout -palvelun. Sen käyttöönotto vaatii sopimuksen osapuolien välille, jotta saadaan käyttöön kauppiastunniste sekä salainen avain palvelun käyttöönottoa varten. OP tuki myös varsinaista koodaustyötä jakamalla tietoa aiemmista heidän keskustelubottien toteutuksista, jossa vastaava toimin-

nallisuus on toteutettu maksujärjestelmää ja Messenger-rajapintaa hyödyntäen. Kokonaan uutena asiana toteutukseen liittyen oli rahankeruulain puitteissa toimiminen eli sijainnin ja paikan varmistaminen maksuun liittyen.

Kohderyhmänä työlle on Uuden Verson kävijät, jotka pääsääntöisesti ovat nuoria aikuisia. Kohderyhmän perusteella voidaan luotettavasti valita toteutusta varten Messenger-alusta, sillä isolla osalla messukävijöistä on valmiina mobiililaitteeseen asennettuna Messenger-sovellus sekä jokin mobiilimaksusovellus, kuten Pivo tai MobilePay.

2.2 Konstruktiivinen metodologia

Työn tutkimusmenetelmä seurasi konstruktiivista metodologiaa, jossa pyritään luomaan ratkaisu oikeaan tosimaailman ongelmaan [2]. Työ keskittyi ratkaisemaan reaalimaailman ongelman, joka tässä tapauksessa oli vaihtoehtoisen maksutavan kehitys kolehdinkeruulle. Ongelma ratkaistiin luomalla mobiilimaksusovellus, innovatiivinen konstruktio, joka on samalla yksi menetelmän ydinkäsitteistä. Metodologian noudattaminen edellyttää yhteistyötä työn toteuttajan tai tutkijan ja käytännön ongelman edustajien kanssa. Yhteistyö tämän diplomityön osalta tapahtui kirjoittajan, joka toimii ohjelmistosuunnittelijana, ja seurakunnan pastorin välillä.

Konstruktiivisen menetelmän valinta tuo käytännön edun toteutusprosessiin liittyen. Sillä voidaan asettaa selkeä tavoite tietyn reaalimaailman ongelman ratkaisuksi. Tässä työssä tavoitteena oli luoda innovatiivinen sovellus, joka on ensimmäinen laatuaan asetetun toimintaympäristön sisällä. Taustatyötä rahankeruuluvan ja kirkkolain vaatimuksista oli tehty kattavasti. Näistä vaatimuksista lisää seuraavassa aliluvussa. Tämä teoreettinen taustatyö ja käytännön tarve uudelle maksutavalle yhdessä muodostavat vahvan pohjan sovelluksen käytännön toteutukselle.

Lopputuloksen testaus ja sen toiminnallisuuden arviointi ovat merkittävä osa konstruktiivista menetelmää [2]. Ohjelmistotuotannon alueella testaus realisoituu kätevästi sekä sovellukselle kirjoitettuina testeinä, että käytännön sovelluksen testauksena. Arvioinnista lisää luvussa 6. MVP periaatteen hyödyntäminen tämän rinnalla toimi myös mainiosti, kun testausta ja tuotteen arviointia suoritettiin iteratiivisesti koko sovelluksen elinkaaren ajan.

2.3 Kolehdinkeruu ja rahankeruulaki

Kolehtia kerätään perinteisesti jumalanpalvelusten aikana kierrättämällä haaveja kävijöiden kautta. Kävijät jättävät omatoimisesti haluamansa määrän käteistä haaviin, josta ne jumalanpalveluksen jälkeen lasketaan ja tilitetään keräyskohteelle. Kuten aliluvussa 2.1 on mainittu, käteisen määrä laskee entisestään vuosittain. Tämän toteutuminen halutaan ennakoida ennen kuin on liian myöhäistä. Jos kirkossa kerätyn kolehdin määrä laskee, niin kannatuskohteet saavat merkittävästi vähemmän kannatusta.

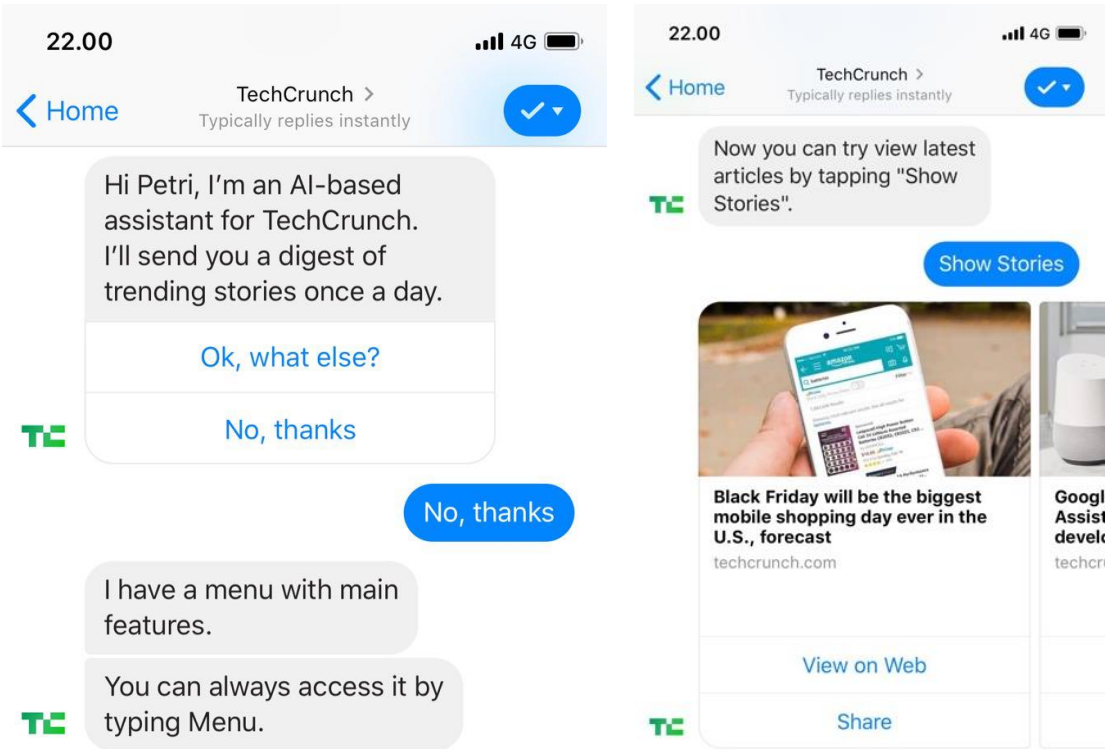
Merkittävin tekninen rajoite sovellukselle on rahankeruulain mukainen toiminta. Eli maksutapahtuma on pystyttävä sitomaan tiettyyn tapahtumaan kirkossa. Tätä asiaa selviteltiin ja riittäväksi tarkkuudeksi todettiin käyttäjän sijaintitiedot laitteelta sekä maksutoiminnallisuuden aukioloaika vain tilaisuuden tai tilaisuuksien välillä. Joskus pidetään kaksi tapahtumaa, joiden välillä tarjotaan kahvit. Maksujen suoritus sinä aikana on sallittua.

3. TOTEUTUSTEKNOLOGIA

Tämä luku sisältää katsauksen työhön valituista toteutusteknologioista. Teknologiat ja käytössä olevat palvelut käydään läpi tarkasti, jotta saadaan hyvä kuva siitä, miten ja miksi mobiilimaksusovellus on toteutettu niitä hyödyntäen. Ensin esitellään Facebook Messenger sovellus ja Facebookin tarjoama avoin rajapinta sen käyttöön. Seuraavaksi keskustelupolun luonti sekä ylläpito ja lopulta vielä esitellään Checkout-maksujärjestelmä sekä sen integraatio verkkopalvelimelle.

3.1 Facebook Messenger rajapinta

Facebook tarjoaa käyttäjilleen Messenger pikaviestintäsovelluksen, jossa voit viestittää joko ystäviesi kanssa, tai nykyään jo erinäisten sivujen tai ryhmien kanssa. Avointa rajapintaa käytetään puhtaasti keskustelubottien ohjelmointiin. Keskustelubotit ovat ohjelmoituja robotteja, jotka vastaavat joko hyödyntäen tekoälyä tai noudattamalla ennalta määrättyä kaavaa, kuten nähdään kuvasta 1. Siinä nähdään TechCrunch nimisen startupin toteuttama keskustelubotti, joka jakaa alustan uusimpia uutisia.



Kuva 1. *TechCrunchin toteuttama keskustelubotti.*

Kuvassa 1 keskustelu on aloitettu TechCrunch sivuston kanssa käyttäen Messenger pikaviestintäsovellusta. Aloittamalla keskustelun keskustelubotti lähettää tervehdyksen ja kysyy haluatko ottaa vastaan push-ilmoituksia. Nämä ilmoitukset summaavat päivän kuumimmat aiheet käyttäjilleen.

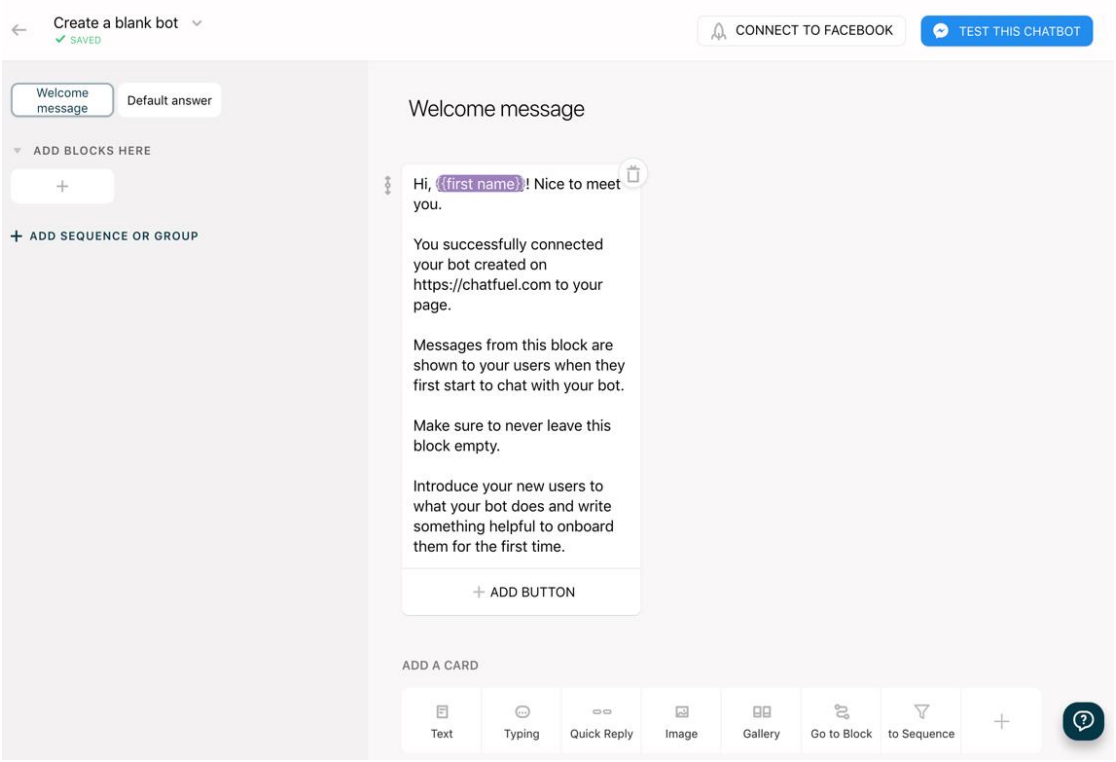
Keskustelussa voidaan hyödyntää rajapinnan tarjoamia erilaisia komponentteja tarpeen mukaisesti. Kuvasta 1 nähdään suoraan muutama suosituimmista komponenteista, joista merkittävin on kuvan oikealla puolella alalaidassa näkyvä karuselli, joka on horisontaalisesti vieritettävissä oleva kokonaisuus. Karusellin jokainen solu sisältää vaihtoehtoisen kuva, otsikon, alaotsikon ja x-määrän toimintakehote painikkeita (engl. CTA). CTA-painikkeita hyödyntämällä käyttäjää voidaan ohjata joko sovelluksen ulkopuolelle tai jatkamaan toiseen keskustelupolkuun saman keskustelun sisällä.

Tätä kyseistä komponenttia käytetään myös mobiilimaksusovelluksessa erityisen paljon, sillä se jäsentää sisällön hyvään kompaktiin ja käyttäjäystävälliseen muotoon. Tämän lisäksi erityinen etu saavutetaan anonymiteettiin liittyen, sillä karusellin linkkejä painamalla ei tule ilmoitusta siitä, mitä ollaan valittu. Tällä voidaan anonymisoida Facebookin päässä tieto käyttäjän antamasta kolehdin määrästä, kun annettava kolehtisumma valitaan ennalta määrätyistä maksuvaihtoehdoista.

3.2 Chatfuel toteuttaa Messenger API:n

Chatfuel on kolmannen osapuolen tuottama palvelu, joka tarjoaa kehitysympäristön keskustelubottien luomiseen. Palvelu tuottaa 46% kaikista Facebook Messenger ympäristön keskusteluboteista [3]. Kehitysympäristön käyttö on mahdollista ilmaiseksi hyväksymällä ilmoitukset käyttäjille, että kyseinen keskustelubotti on rakennettu käyttämällä heidän palveluaan.

Chatfuel tarjoaa graafisen käyttöliittymän, jonka avulla voidaan rakentaa keskustelukulku raahaamalla rajapinnan tarjoamia komponentteja tiettyyn järjestykseen, kuten nähdään kuvassa 2. Oikeassa alakulmassa on lista eri komponenteista, jotka sisäisesti toteuttavat Facebookin rajapinnan vaatimukset komponenteiltaan.



Kuva 2. *Chatfuel palvelun tarjoama graafinen käyttöliittymä keskustelukulun luomiseksi.*

Vertaamalla kuvaa 2 ja ohjelmaa 1 nähdään suoraan merkittävät hyödyt, jotka graafinen käyttöliittymä tarjoaa keskustelukulun luomiseen. Erityisesti komponenttien monimutkaistuessa niiden luomisen ja ylläpidon kannalta käyttöliittymä on ylivoimainen tapa, kun keskustelubotin ylläpito vieritetään vähemmän tekniselle käyttäjälle.

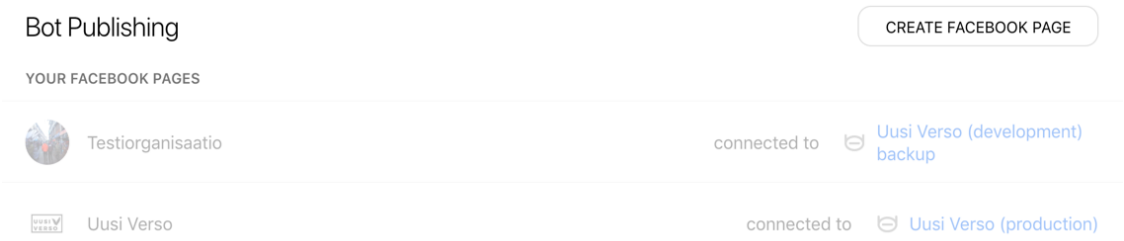
```
"greeting":[
  {
    "locale":"default",
    "text":"Hello {{user_first_name}}!"
  }
]
```

Ohjelma 1. *Tervehdysviestin luominen Facebook Messenger alustalle.*

Kuvassa 2 näkyy oikeassa alareunassa navigaatio, josta käyttäjä voi raahata komponentteja ja rakentaa niillä haluamansa keskustelukulun. Keskustelukulun lisäksi palvelu tarjoaa kaksi erityistarkoituksen omaavaa viestiä. Ensimmäinen niistä on tervetuloviesti, joka näkyy kuvassa yllä. Viesti lähetetään käyttäjälle välittömästi, kun keskustelu aloitetaan botin kanssa. Tässä tilanteessa on syytä kertoa, että kyseessä on keskustelubotti, eikä toisella puolella ole välttämättä ihmistä vastaamassa kysymyksiin. Viesti voidaan personoida käyttämällä hyväksi rajapinnan tarjoamia käyttäjätunnisteita, kuten kuvassa on käytössä etunimi. Jälkimmäinen erityisviesti on ns. oletusvastaus (engl. default answer), joka lähetään aina kun käyttäjän botille kirjoittamaa viestiä ei osata tulkita. Siinä tilanteessa

botti vastaa aina oletusvastauksella. Siinä voidaan esimerkiksi ohjata käyttäjä keskustelupolun alkuun ja pahoitella, että ei osata vastata viestiin.

Botin luominen kehitysympäristön avulla aloitetaan yhdistämällä oma Facebook-ryhmä tai -sivu palveluun. Tämä toiminto on yksinkertaista suorittaa ja se voidaan tehdä käyttöliittymän kautta, kuten nähdään kuvassa 3.



Kuva 3. *Keskustelubotin yhdistäminen Facebook ryhmään tai sivustoon.*

Chatfuel on korvaamaton palvelu mobiilimaksusovellukselle, sillä ilman graafista käyttöliittymää, keskustelupolun rakentaminen komponentti komponentilta olisi suunnattoman suuritöinen tehtävä sen tuomaan hyötyyn nähden. Palvelun ilmaisversio rajoittaa käytön tiettyyn komponenttilistauksen käyttöön sekä asettaa reunaehdot niiden käytöstä eri tapauksissa. Jos keskustelubotti ohjelmoitaisiin kokonaan itse, voitaisiin komponenttien toteutuksen suhteen välttää niille asetetut rajaukset. Toisaalta samalla komponenttien ylläpito ja keskustelupolun muokkaaminen olisi lähes mahdoton tehtävä seurakunnan työntekijöille. Tavoitteenahan on siirtää botin ylläpito pois kehittäjältä.

Ylläpitotyön siirtäminen varsinaiselle käyttäjäryhmälle seurakunnan piiriin on tärkeää, sillä keskustelubottiin voidaan lisätä myös paljon muuta sisältöä, kuten linkkejä blogeihin, SoundCloud sisältöihin sekä moneen muuhun palveluun potentiaalisesti. SoundCloudiin tallennetaan kolehdinkeruutilaisuudessa eli jumalanpalveluksen aikana opetettu saarna. Näille eri palveluista tuoduille sisällöille Chatfuel tarjoaa valmiin toiminnallisuuden, jonka voi upottaa käyttöön keskustelupolun varrelle. Mahdollisia muutoksia bottiin on siis vähintään viikoittain. Myös eri tilaisuuksia varten voidaan lisätä maksupainikkeita, joilla voidaan suorittaa osallistuminen tiettyyn tilaisuuteen tai suorittaa mikromaksuja mihin tahansa tilanteeseen.

3.3 Checkout-maksujärjestelmä

Checkout-maksujärjestelmä on Checkout Finland Oy:n maksunvälitys palvelu, joka on osa OP Ryhmää. Järjestelmä tarjoaa vaivattoman maksunvälityksen sekä kauppiaan, että asiakkaan näkökulmasta. Checkout Finland Oy lupaa verkkosivuillaan tarjoavansa yhden sopimuksen avulla koko maksunvälityspalvelun. He hoitavat kaiken viestinnän pankkeihin ja luottolaitoksiin.

Maksujärjestelmä tarjoaa erittäin kattavan teknisen materiaalin, tai rajapintakuvauksen, joka sisältää myös erilaisten valmiiden ohjelmistopakettien integraatio kokonaisuuksia. Näiden ohjelmistopakettien lisäksi tarjolla on myös Javalla, PHP:lla ja Python toteutetut kirjastot, joita voi kätevästi hyödyntää omaan toteutukseen. OP:lta annettiin myös vinkki näiden ohjelmapakettien ja kirjastojen ulkopuoliseen avoimen lähdekoodin Node.js:llä kirjoitettuun kirjastoon [4], jota voisi hyödyntää mahdollisesti omaan toteutukseen.

Node.js oli sovelluksen toteuttajalle aiemmalta kokemukselta lähes tuntematon ympäristö. Kauniston tekemä kirjasto [4], joka sisältää kaikki tarvittavat työkalut ja toiminnot Checkoutin rajapinnan käyttöön, nopeutti ensimmäisen prototyypin valmistumista huomattavasti. Näin ollen fokus voitiin suunnata varsinaisen maksujärjestelmän käyttöönottoon, testaukseen ja rajapinnan ymmärtämiseen.

Työn kirjoitus hetkellä rajapinnasta on tullut jo uusi versio, mutta mobiilimaksusovellus on integroitu vanhaan ”legacy” versioon. Kuten aiemmin mainittiin, tekninen dokumentaatio on kattava ja sisältää erittäin paljon esimerkkejä sekä ohjeita useiden sisäisten rajapintojen käyttöön. Rajapintakuvaus sisältää selitteiden lisäksi lähes jokaiseen kohtaan selkeän yksiselitteisen ohjelmakoodin, joka on toteutettu esimerkin mukaisesti PHP:lla, kuten kuvassa 4.

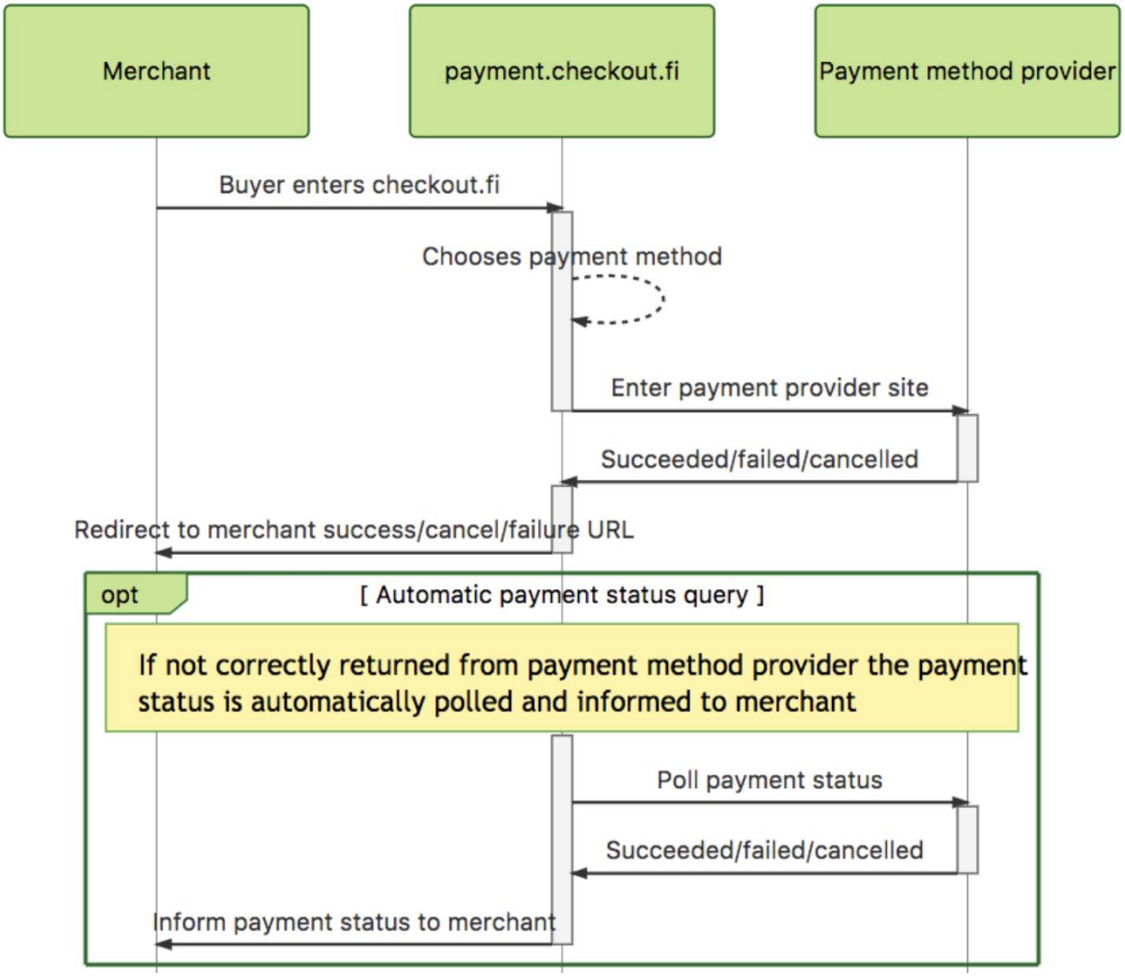
#	Description	Name	Value	Format	Required
1	Payment version. Always “0001”.	VERSION	“0001”	AN 4	Yes
2	Unique identifier for the payment in the context of the merchant.	STAMP	unique_id	AN 20	Yes
3	Payment amount in cents.	AMOUNT	0	N 8	Yes
4	Payment reference number for the payment from the merchant.	REFERENCE	standard_reference	AN 20	Yes
5	Payment message from the Merchant to the buyer	MESSAGE	“Hi, thanks for shopping.”	AN 1000	No
6	Payment language. Options currently are: “FI”, “EN”, “SE”.	LANGUAGE	“FI”	AN 2	No
7	Merchant Id. Given identifier for the merchant (you get this from Checkout Finland), for testing: 375917	MERCHANT	375917	AN 20	Yes
8	Return callback. Called when the payment is successfully paid.	RETURN		AN 300	Yes

```php
<?php

// Order information
$orderData = array(
    'STAMP'         => time(), // Unique timestamp
    'REFERENCE'     => '12344',
    'MESSAGE'       => "Furniture materials\nWoodworking tools",
    'RETURN'        => 'http://example.com/return',
    'CANCEL'        => 'http://exaple.com/cancel',
    'AMOUNT'        => '1000', // Price in cents
    'DELIVERY_DATE' => date('Ymd'),
    'FIRSTNAME'     => 'Matti',
    'FAMILYNAME'    => 'Meikäläinen',
    'ADDRESS'       => "Ääkköstie 5 b 3\nGround floor",
    'POSTCODE'      => '33100',
    'POSTOFFICE'    => 'Tampere',
    'EMAIL'         => 'support@checkout.fi',
    'PHONE'         => '0800 552 010'
);

// Find the client class definition above
$client = new Checkout(375917, 'SAIPPUAKAUPPIAS');
$xmlString = $client->getPaymentButtons($orderData);
$xml = simplexml_load_string($xmlString);
var_dump($xml);
```

Kuva 4. *Kuvakaappaus kattavasta teknisen dokumentaation rajapintakuvauksesta lähdekoodeineen.*

Rajapintaa toimii siten, että ensin asiakas käynnistää maksuportaalin verkkopalvelun kautta, johon maksujärjestelmä on integroitu. Summa ja maksutapa (MobilePay, Pivo tai Siirto) valitaan etukäteen Messengerissä ja välitetään parametreina verkkopalvelimelle. Niiden avulla alustetaan maksun tiedot ja uudelleenohjataan asiakas suoraan alustetun maksun kanssa valitun maksutavan sovellukseen. Tässä vaiheessa asiakas suorittaa maksun. Jos maksu onnistui, asiakas ohjataan rajapinnalle toimitettuun onnistuneen maksun osoitteeseen ja mikäli ei, epäonnistuneen maksun osoitteeseen. Jos jokin menee odottamattomasti pieleen, ja asiakasta ei ohjata maksutavan toteuttavan sovelluksen puolesta

takaisin, voidaan maksun tila kysellä siihen tarkoitetulta rajapinnalta (Polling API). Kuvassa 5 nähdään kuinka rajapinta toimii kauppiaan, asiakkaan ja maksujärjestelmän suhteen. Siitä nähdään kuinka kyselyrajapintaa voidaan hyödyntää maksun onnistumisen varmistamiseksi.



Kuva 5. *Rajapintakuvaus Checkout Finlandin teknisestä dokumentaatiosta, jossa asiakkaan ja maksun kulku kuvattu [5].*

Kun sopimusasiat saadaan ratkottua ja voimassa oleva sopimus on tehty, rajapinnan integrointiin käytetään kauppiaalle toimitettuja kauppiastunnus ja kauppias-salausavain paria. Näiden avulla voidaan yksilöidä tietyn sopimuksen tiedot järjestelmään ja osoittaa maksut oikeaan osoitteeseen. Koska sopimuksen teossa voi kestää ja sen teko voi sisältää aikaa vievää neuvottelua, tekninen dokumentaatio tarjoaa testitunnukset, joilla voidaan aloittaa maksuportaalin integraation toteutus.

Näillä tiedoin voidaan jo toteuttaa koko integraatio sekä testata maksujen onnistumista ja epäonnistumista tai keskeytymistä suoraan verkkopankkiin asti. Useat verkkopankit tarjoavat myös testitunnukset, joilla voidaan kirjautua sisään verkkopankkiin ja testata koko järjestelmän alusta loppuun. Lopulta voidaan vain lisätä todelliset kauppiastunnisteet ja tehdä viimeinen testaus.

Kolmannen osapuolen rajapintojen käyttöönotto usein vaatii tunnistetietojen käyttöä, jolla voidaan tunnistaa käyttäjä. Tässä tapauksessa edellä mainittu kauppiastunnus- ja salausavain-pari toimii pääsynä rajapintaan. Nämä tiedot on syytä pitää ohjelmakoodissa piilossa. On järkevää, että tunnistautumiseen tarvittavat tokenit ja avaimet eivät kulje muun ohjelmakoodin mukana esimerkiksi versionhallintajärjestelmään. Erityisesti tietysti silloin, jos kyseessä on julkisen lähdekoodin projekti.

Ongelmaan löytyy ratkaisu. Usein käytetään ympäristömuuttujia asettamaan edellä mainitut salausavaimet tai muut tarvittavat tunnisteet. Otimme käyttöön työkalun nimeltä dotenv (.env), jolla voidaan lisätä ympäristömuuttujat .env nimiseen tiedostoon projektin juurihakemistoon. Tämä tiedosto asetetaan .gitignore nimiseen tiedostoon, jonka avulla käytössä oleva versionhallintajärjestelmä jättää huomiotta kyseisen tiedoston ja siten estää sen kulun versionhallintajärjestelmään. Ympäristömuuttujat saadaan sovelluksessa käyttöön ohjelman 2 kuvaamalla tavalla, josta nähdään työkalun toiminnallisuus.

```
/**
 * Import .env and instantiate an API object with credentials.
 */
require('dotenv').config();

const CheckoutApi = require('checkout-api');

const checkout = new CheckoutApi({
  merchantId:          process.env.MERCHANT_ID,
  merchantSecret:      process.env.MERCHANT_SECRET,
  baseUrl:             process.env.BASE_URL,
  allowSmallPurchases: false
});

```

Ohjelma 2. *Ympäristömuuttujien käyttö dotenv työkalun avulla.*

Nodella se lisää muuttujat process-objektiin, joka sisältää sovelluksen globaalin tilan. Tällä tavalla saadaan erotettua konfigurointi varsinaisesta ohjelmakoodista.

3.4 Verkkopalvelin

Checkout-maksujärjestelmä on integroitu verkkopalvelimelle, jota ajetaan DigitalOcean nimisen pilvipalvelutarjoajan avulla. DigitalOcean on pilvilaskentaa tarjoava yritys, joka on kehittänyt palvelunsa ohjelmistokehittäjän näkökulmasta. Palvelun avulla kehittäjän on yksinkertaista ja suoraviivaista rakentaa palvelininfrastruktuuri käyttöliittymän ja heidän tarjoaman rajapinnan kautta.

Jos sovelluksen käyttäjämäärä moninkertaistuisi, alustalla voidaan suorittaa skaalaus nopeasti asettamalla useita virtuaalikoneita samasta instanssista. Tämä on myös erityisesti mahdollista, sillä palvelin toimii tilattomasti, joka tarkoittaa sitä, että kaikki palvelimen

toiminta perustuu vain ja ainoastaan kutsun mukana lähetettyihin parametreihin. Parametrien perusteella vastaus voidaan luoda aina identtisesti samanlaiseksi. Käytännössä tämä tarkoittaa sitä, että samaa sovellusta voidaan ajaa tarvittaessa samanaikaisesti vaikka tuhannella eri virtuaalikoneella, eivätkä muut instanssit palvelusta tarvitse tietää toisistaan mitään.

Virtuaalikoneella tarkoitetaan ohjelmallisesti toteutettua tietokonetta, jota ajetaan ja siihen voidaan olla etäyhteydessä omalta fyysiseltä tietokoneelta. Tässä tapauksessa DigitalOcean pilvipalveluntarjoajana tarjoaa tarvittavan määrän virtuaalikoneita asiakkaiden käyttöön, joita ajetaan fyysisesti heidän omalla laitteistollaan datakeskuksissa. Tämä palvelumuoto, jossa asiakkaalle tarjotaan virtuaalikoneita on tehokas molempien osapuolien osalta [6]. Asiakkaiden ei tarvitse itse ylläpitää fyysistä laiteinfrastruktuuria, joka voi olla hyvinkin kallista, jos laitteet eivät ole käyttöasteeltaan korkeita. Palveluntarjoaja taas pystyy tällä tavoin kasvattamaan laitteiden käyttöastetta, kun yhden fyysisen koneen resursseja voidaan virtualisoida useamman asiakkaan tarpeiden mukaan.

DigitalOcean alustalla yksittäistä laskenta-alustaa kutsutaan termillä Droplet (suomeksi pieni pisara). Dropletit ovat enemmän kuin pelkästään virtuaalikoneita, sillä ne ovat skaalautuvia laskenta-alustoja. Niitä voidaan kustomoida omien tarpeiden mukaan riippuen kuinka suuret tarpeet ajettavalla sovelluksella on esimerkiksi koneen muistin, välimuistin ja suorituskyvyn (prosessorien määrää, tehokkuus) suhteen. Niillä voidaan ajaa valmiiksi tuotteistettuja Linux distroja eli ohjelmakokoelmia, kuten Ubuntua, Debiania tai muuta vastaavia.

DigitalOceanin tarjoama palvelu voidaan luokitella IaaS-palveluksi (Infrastructure as a Service). Sitä voidaan kutsua IaaS-palveluntarjoajaksi, sillä se tarjoaa webin välityksellä virtuaalikoneita käyttäjien tarpeiden mukaan [6]. IaaS-palvelussa vastuu tietoturvapäivityksistä ja muista sovellustason asioista jää kehittäjälle. Kehittäjän vastuulla on myös valita käyttöjärjestelmä ja ohjelmistopaketti, jota virtuaalikoneella ajetaan sekä valitun kokonaisuuden konfigurointi. Palveluntarjoaja hoitaa fyysisistä laitteistosta, jolla virtuaalikoneita ylläpidetään ja hoitaa myös tallennustilaan, muistiin ja suorituskykyyn tarjonnan.

Yhdellä Dropletilla ajetaan siis yhtä distroa, joka voidaan kustomoida sovelluksen tarpeiden mukaan. Näiden lisäksi voidaan asettaa automaattinen varmuuskopiointi, valita minkä alueen datakeskukseen haluaa sovelluksen ajoon sekä muita yksikohtaisia konfiguraatioita. Mobiilimaksusovellusta varten valittiin alustavasti mahdollisimman edullinen palvelinkokonaisuus, sillä tarpeen kasvaessa, sitä on helppo skaalata suoraan käyttöliittymästä. Näin ei makseta turhasta ennen kuin tarve kasvaa.

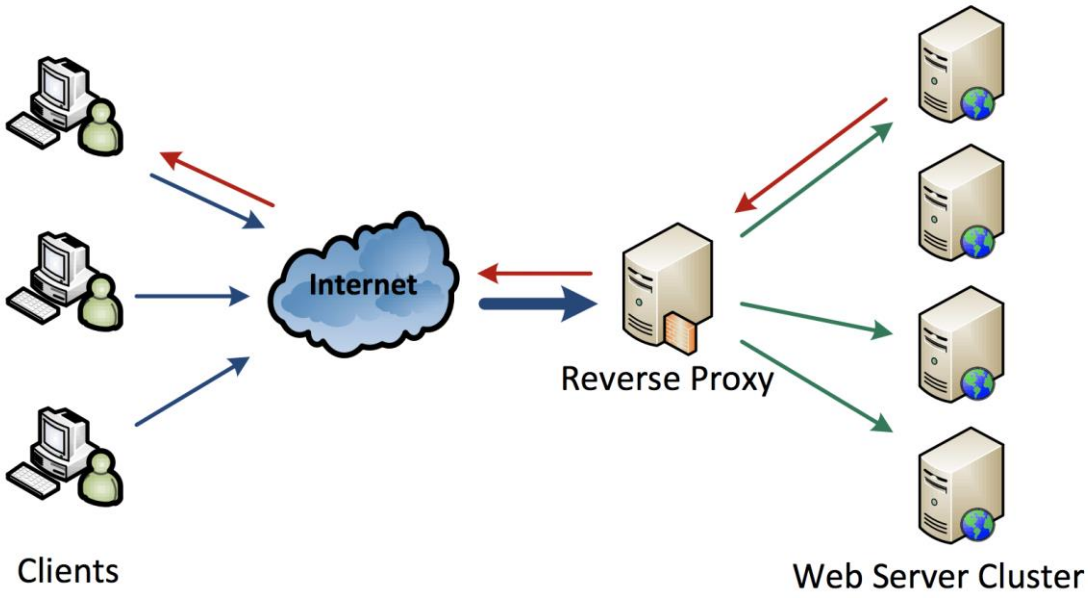
Distroksi valikoitui Ubuntun 16.04.3 x64 versio, joka sisältää 512 MB välimuistia, 20 GB kovalevytilaa, ja palvelinkeskus sijaitsee Frankfurtissa, Saksassa. Ubuntu on avoimen lähdekoodin käyttöjärjestelmä, joka on maailman suosituin käyttöjärjestelmä julkisten

pilvipalveluiden tarjoajien keskuudessa [7]. Ubuntulle asennettiin Nginx niminen verkkopalvelin, joka on puolestaan yksi suosituimmista verkkopalveluista Linux-alustalla.

Sovellusta varten Nginxiä ajetaan käänteisenä välityspalvelimena (engl. reverse proxy). Käänteisen välityspalvelimen toiminnallisuus lisää palvelimen turvallisuutta ja toimii samalla kuormantasaajana. Sillä voidaan ottaa kiinni palvelimelle osoitetut pyynnöt ja ohjata ne oikein paikallisesti Node.js:llä pyörivään backendiin. Tämä lisää turvallisuutta, sillä palvelimen resurssit ei ole suoraan saavutettavissa ulkoapäin, vaan palvelinta ajetaan turvallisesti sisäverkossa.

Ulkoapäin palvelimelle saa yhteyden Dropletille asetettavasta verkkotunnuksesta, esimerkiksi muodossa *https://example.com*. Sen voi hankkia verkkotunnusteita myyvältä palvelulta mistä tahansa. Asetuksiin syötetään verkkotunnuksen A Record, jonka avulla kartoitetaan palvelimen IP-osoite tiettyyn verkkotunnukseen. Näin ollen ei tarvitse käyttää IP-osoitetta, että saadaan yhteys palvelimeen, vaan voidaan hyödyntää ihmisläheisempää luettavaa osoitetta. Esimerkiksi IP-osoitteelle *http://178.62.222.133* asetetaan A Recordin avulla luettavampi ja muistettavampi osoite *http://petrisandstrom.com*. Verkkotunnus ei vielä sellaisenaan vastaa, vaan toiminnan edellytyksenä on nimipalvelimien määrittely. Nimipalvelimet asetettiin DigitalOceanin hallinnoimiin palvelimiin ja tämän jälkeen tarvitsee odottaa noin vuorokausi, että uuden verkkotunnuksen tiedot on välitetty nimipalvelujärjestelmään.

Kuormantasaajana käänteinen välityspalvelin toimii siten, että samaan verkkotunnukseen lähetettyjä kutsuja voidaan ohjata useammalle verkkopalvelimelle riippuen kuorman määrästä ja palvelimen rasitustilasta. Asiaa voidaan ajatella liikenteenohjaajana, joka maksimoi suorituskyvyn ja kapasiteetin jakamalla kaiken liikenteen tasaisesti eri palvelimien välillä, kuten kuvassa 6. Kuormantasaaja lisää myös luonnollisesti palvelun saatavuutta, jos yksi palvelin kaatuu tai päätyy tavalla tai toisella virhetilaan, ohjataan liikenne muihin toimiviin instansseihin.



Kuva 6. *Asiakaspään kyselyjen reititys palvelimille [8].*

Nginxin konfigurointi aloitettiin muokkaamalla sen konfigurointitiedostoa, joka sijaitsee polussa `/etc/nginx/sites-available/default`. Ensimmäiseksi haluttiin saada HTTP2 tuki kytkettyä päälle, ja tämä edellyttää selaimissa HTTPS-protokollan käyttöönottoa. Lisää HTTPS-protokollasta myöhemmin tässä luvussa. Kaikki HTTP-protokollan kutsut täytyy siis ohjata portista 80 porttiin 443. Tämä voidaan tehdä ohjelman 3 mukaisesti edellä mainittuun tiedostoon.

```
# HTTP - redirect all traffic to HTTPS
server {
  listen 80;
  listen [::]:80 default_server ipv6only=on;
  return 301 https://$host$request_uri;
}
```

Ohjelma 3. *Nginx konfiguraatio HTTP-protokollan liikenteen uudelleenohjaamiseksi porttiin 443, joka vastaanottaa HTTPS-protokollan viestit [9].*

Corbel et al. mukaan verkkopalveluiden keskimääräinen koko on tasaisesti nousujohteinen ja kaksinkertaistunut vuosien 2012 ja 2015 välillä. Suhteellisen uusi HTTP2-protokolla pyrkii ratkaisemaan edellisen sukupolven HTTP1.1-teknologian puutteita, kuten head-of-line eston. Esto rajoittaa selaimen samanaikaisien yhteyksien määrää. Selaimet pystyvät suorittamaan maksimissaan 6 yhtäaikaista TCP-yhteyttä. Uudet yhteydet näiden jälkeen joutuvat odottamaan, että jokin aiemmasta kuudesta yhteydestä lopetetaan. Hyvä puoli uuden protokollan käytössä on se, että jos ympäristö ei tue protokollaa, se on taaksepäin yhteensopiva ja hyödyntää silloin vanhempaa HTTP1.1-protokollaa. [10]

Seuraavassa ohjelmassa 4 nähdään konfigurointi, jolla käynnistetään tuki HTTP2-protokollalle ja avataan portti 443 käyttöön SSL salatulle liikenteelle rivillä 4. Kuudennella rivillä asetetaan palvelimelle verkkotunnus.

```
# HTTPS - proxy all requests to the Node app
server {
  # Enable HTTP/2
  listen 443 ssl http2;
  listen [::]:443 ssl http2;
  server_name <INSERT_URL>;

  # Use the Let's Encrypt certificates
  ssl_certificate /etc/letsencrypt/live/<INSERT_URL>/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/<INSERT_URL>/privkey.pem;

  # Include the SSL configuration from cipherli.st
  include snippets/ssl-params.conf;

  location / {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Nginx-Proxy true;
    proxy_pass http://localhost:3000/;
    proxy_ssl_session_reuse off;
    proxy_set_header Host $http_host;
    proxy_cache_bypass $http_upgrade;
    proxy_redirect off;
  }
}
```

Ohjelma 4. *Nginx konfiguraatio sertifikaatin, SSL-yhteyden ja käänteisen välityspalvelimen asetukseen [9].*

Samalla lisättiin asetuksiin location-lohkoon riviltä 15 alkaen konfigurointi, jolla käänteinen välityspalvelin saadaan toimimaan ja ohjaamaan liikenteen paikalliselle Node-palvelimelle.

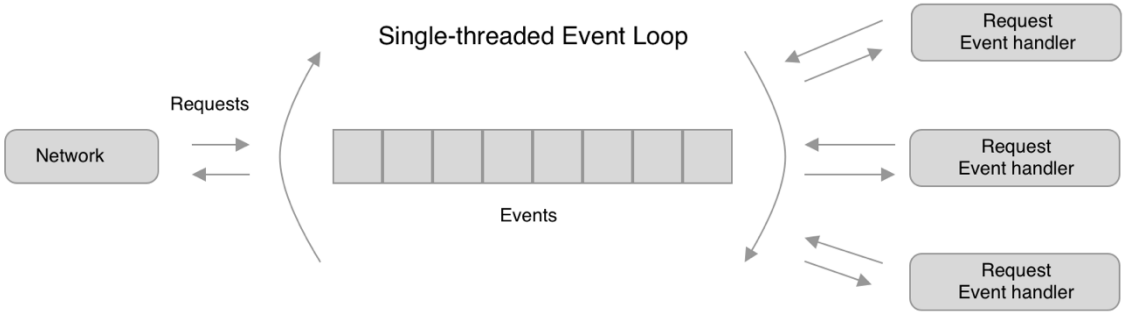
Nginxin käänteisen välityspalvelin voi myös estää palvelunestohyökkäyksen (DDoS). Palvelunestohyökkäys tarkoittaa sitä, että yhteen verkkotunnukseen johdetaan erittäin paljon liikennettä ja palvelin ruuhkautuu, kun ei pysty käsittelemään pyyntöjen määrää. Tämä voidaan estää esimerkiksi mustalistaamalla tiettyjä IP-osoitteita, jotka lähettävät pyyntöjä jatkuvasti samaan resurssiin ja pyrkivät aiheuttaa palvelimelle ruuhkaa. Toinen vaihtoehto on rajoittaa käyttäjien yhtäaikaisten yhteyksien määrää palvelimelle.

Palvelimen ohjelmointikieleksi valittiin JavaScript, sillä sovelluksen kehittäjä on kokenut asiakaspään (engl. frontend) ohjelmistosuunnittelija. Asiakaspää on tyypillisesti tapahtumakeskeinen asynkroninen ympäristö, johon JavaScript soveltuu täydellisesti. Selain on se ympäristö, josta puhutaan kun on kyse asiakaspään ohjelmoinnista ja tapahtumat siellä tarkoittavat usein käyttäjän interaktiosta aiheutuneita tapahtumia, esimerkiksi käyttöliittymässä painikkeen klikkausta. Kieli on viime vuosina yleistynyt myös palvelinpään

(engl. backend) puolella, jossa Node.js:n suosio on kasvanut merkittävästi. Projekti osoittautui hyväksi mahdollisuudeksi opetella samalla backend-ohjelmointia aiemmin tutulla kielellä.

Paikallisesti ajetaan Node.js:llä kirjoitettua palvelinta, joka toteuttaa integraation Checkout-maksujärjestelmään. Node.js on avoimen lähdekoodin JavaScriptillä ohjelmoitava ajonaikainen ympäristö [11], joka perustuu Googlen V8 moottoriin. Node.js on sisäisesti kirjoitettu C/C++-kielellä ja sen päätavoitteina on suorituskyky sekä pitkäkestoisten palvelinprosessien tuki. Koska kyseessä on JavaSript-ympäristö, kyseessä on asynkroninen yksisäikeinen prosessimalli, joka eroaa palvelinpuolella perinteisestä monisäikeisestä eli rinnakkaisuutta tukevasta järjestelmästä merkittävästi.

JavaScript toteuttaa näennäisen rinnakkaisuuden prosessilla, jonka keskiössä on tapahtuma-silmukka (engl. event loop), kuva 7. Sen tehtävänä on ajoittaa ja suorittaa eri tehtävien toteutuminen. Toinen merkittävä toiminnallisuus tämän tavan toimivuudelle on tapahtumien callback toimintamalli. Tämä käytännössä tarkoittaa sitä, että tapahtumien (usein käyttäjän interaktion tai toisen koodin suorituksen myötä) laukaistaan callback, joka ohjaa ohjelman suorituksen haluttuun pisteeseen ehdon täytyttyä.

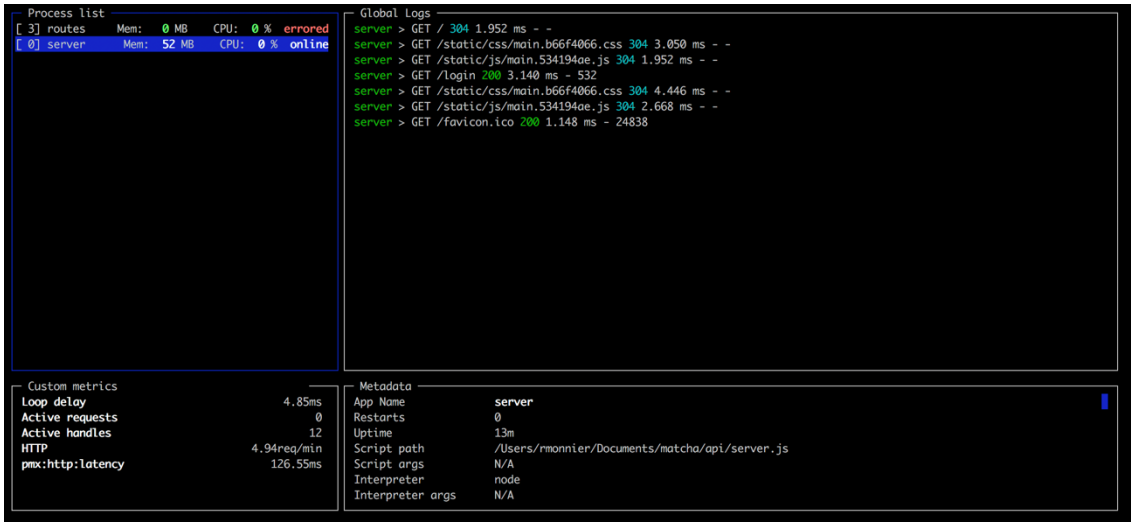


Kuva 7. *Yksisäikeinen tapahtumakeskeinen prosessi.*

Nodea voidaan ajaa suoraan komentoriviltä, jolloin virhetilanteessa normaalisti JavaScript ohjelma kaatuu eikä pysty itse suorittamaan palautumista tai uudelleen käynnistymistä. Koska virhetilanteita voi sattua, ongelma päätettiin ratkaista hyödyntämällä PM2:sta, joka on Noden prosessinhallintatyökalu. Sen avulla voidaan skaalata saman palvelimen kaikki prosessorit ajamaan omia instanssejaan palvelimesta ilman mitään muutoksia ohjelmaan. Tätä toiminnallisuutta varten on ehdotonta, että palvelin toimii tilattomana eikä paikallisesti hoideta mitään tilankäsittelyä, sillä prosessit eivät ole tietoisia toistensa jakamista resursseista.

Käynnistämällä palvelinohjelman suoraan PM2 työkalun avulla, käynnistetään oma prosessi palvelimella, joka ylläpitää ohjelman suoritusta automaattisesti. Jos ohjelma kaatuu, se käynnistetään automaattisesti uudestaan. Työkalun avulla voidaan myös suorittaa prosessitason monitorointia, josta saadaan tietoon esimerkiksi sovelluksen sen hetkinen tila,

prosessorin ja muistin käyttöasteet sekä viimeisimmät palvelimelle lähetetyt kutsut, kuten kuvassa 8.



Kuva 8. *Kuvakaappaus monitorointityökalusta [12].*

Nyt palvelin hallitsee kuormantasauksen, sisältää jonkin verran tietoturvaa ja prosessia ylläpidetään prosessinhallintatyökalun avulla. Seuraavaksi on syytä asettaa TLS/SSL-tason salaus. Tähän käytetään ilmaista varmenteen myöntäjää (engl. Certificate Authority, CA) palvelun Let's Encrypt avulla. Varmenteen avulla saadaan käyttöön salausprotokolla turvalliselle tietoliikenteelle HTTPS-protokollalla.

HTTPS-protokollan avulla siirrettävä tieto salataan ennen lähetystä ja se kulkee verkon yli salattuna. Protokolla aloitetaan yhdistämällä molempien päiden välinen yhteys ja luomalla luottamus osapuolten välille TLS-kättelyllä (engl. TLS handshake). Luoton varmistamiseksi palvelin lähettää selaimelle oman varmenteen, jonka asiakaspää varmistaa. Mikäli validointi onnistuu, voidaan osapuolien välille muodostaa salattu yhteys kommunikaatiota varten.

Let's Encrypt toimii varmenteen myöntäjänä. Certbot-sovelluksen avulla varmenteen hankinta ja ylläpito hoituu automaattisesti, kun käytössä on Nginx [13]. Kuvasta 9 nähdään esimerkki Certbot-sovelluksen käytöstä. Varmenteen automaattinen uusinta on myös mahdollista hoitaa sovelluksella asettamalla automaattinen uusinta, joka ajetaan kaksi kertaa päivässä. Varmenne uusitaan, jos sen voimassaoloaika on alle 30 päivää.

```
# sudo certbot --nginx
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator nginx, Installer nginx

Which names would you like to activate HTTPS for?
-------------------------------------------------------------------------------
1: example.com
2: www.example.com
-------------------------------------------------------------------------------
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel):
```

Kuva 9. *Kuvakaappaus Certbotin käytöstä [13].*

Salatun yhteyden muodostaminen nykypäivänä on jo melko yleistä, mutta erityisen tärkeää tässä tapauksessa sen muodostamiseksi on, että sovelluksen ja verkkopalvelinta käytetään rahaliikenteen aloittamiseksi. Vaikka kolmannen osapuolen Checkout-maksujärjestelmä hoitaa varsinaisen rahaliikenteen ja tunnistautumisen, voidaan lisätä sovelluksen luotettavuutta salaamalla kaikki liikenne koko sovelluksen käytön suhteen.

4. TOTEUTUS

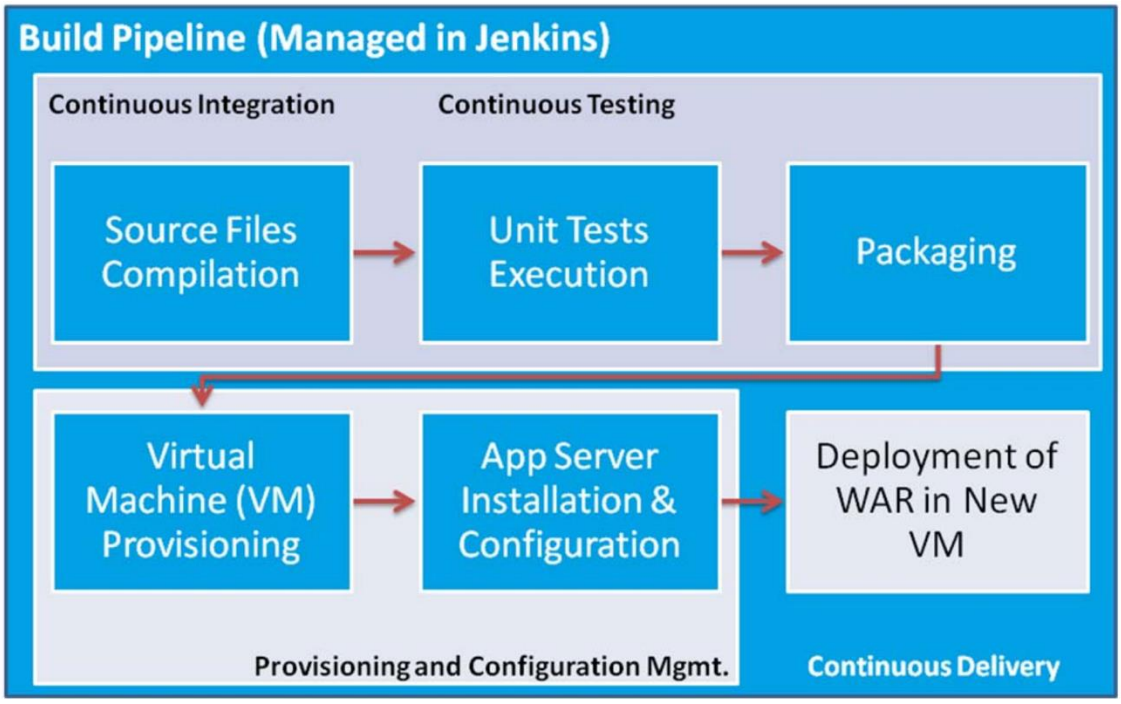
Tässä luvussa esitellään aluksi toteutusprosessiin käytetyt menetelmät. Tämän jälkeen tarkastellaan käyttöliittymää ja esitellään miten aiemmissa luvuissa selitetyt teknologiat sulautuvat sovelluksen käyttöliittymään. Keskustelukulku esitellään kokonaisuudessaan samalla antaen hyvän kuvauksen miten koko sovellus käytännössä toimii. Keskustelukulusta nähdään myös kolehtimaksun lisäksi tehdyt ominaisuudet sovellukseen, kuten blogitekstien integraatiot. Luvun lopussa on katsaus sovelluksen testaukseen liittyen.

4.1 Toteutusprosessi

Projekti aloitettiin mahdollisimman kevyellä suunnitelmalla ilman tarkemmin asetettua aikataulua. Tarkoituksena oli noudattaa iteratiivista MVP periaatetta, jossa toiminnollisuuksia toteutetaan nopeasti. Samalla pyritään rakentamaan prototyyppiversio usein ja validoida sen tarpeellisuus jatkokehitystä varten. Nopealla palautteella kehitystyön laatua pyrittiin parantamaan, ja samalla vähentämään turhiin tai muutoksia vaativiin toimintoihin kuluvaa aikaa hyvissä ajoin prosessia. Pääasiallinen tavoite menetelmällä on tuottaa lisäarvoa sovellukselle mahdollisimman pienellä vaivannäöllä. [14]

MVP periaate on alkujaan tuotu yleiseen tietoisuuteen Eric Riesin Lean Startup metodologiasta. Periaate kuvataan olevan jatkuvasti kehittyvä konsepti. Ries kuvaa sitä uutena versiona tuotteesta, jonka avulla voidaan kerätä mahdollisimman paljon validoitua oppimista asiakkaista näkemällä mahdollisimman pientä vaivaa. [15]

Alkuperäisenä tavoitteena projektin osalta oli myös tutustua ja opetella CI- ja CD-putken (Continuous Integration & Continuous Delivery) käyttö. Sonin mukaan [16] jatkuva integraatio (CI) tarkoittaa tapaa, jolla lähdekoodin tasoa pyritään parantamaan heti projektin elinkaaren alusta alkaen. Käytännön mukaan lähdekoodia julkaistaan riittävän vilkkaaseen tahtiin, esimerkiksi vähintään päivittäin. Jos kyseessä on tiimi, niin sen koosta riippuen lähdekoodin integraatioiden määrä toisiinsa on useita päivässä. Soni jatkaa, jatkuvan toimituksen (CD) olevan ohjelmistotuotannon menetelmä, jossa rakennetaan eräänlainen julkaisuputki (engl. deployment pipeline). Sen läpi kulkee uudet versiot sovelluksesta, ja ne julkaistaan tuotantoon välittömästi. Kuvassa 10 nähdään esimerkki CI/CD-putkesta ja siitä mitä koko prosessi voisi sisältää. [16]



Kuva 10. *Esimerkki jatkuvan integroinnin ja jatkuvan toimituksen metodien käyttämästä julkaisuputkesta [16].*

Yllä olevassa kuvassa aluksi käännetään lähdekoodi, sen jälkeen ajetaan kaikki projektin yksikkötestit varmistaaksemme ettei ohjelmistovirheitä kulje tuotantoon huomaamatta. Seuraavaksi pakataan kokonaisuus tuotantoa varten ja provisioidaan virtuaalikoneen resurssit. Lopulta julkaistaan virtuaalikoneelle uudet muutokset.

Edellä mainittua julkaisuputken toteuttamista kutsutaan usein termillä DevOps, joka tulee englanninkielen sanoista Development-Operations. DevOps sisältää myös paljon muuta, ja tämä olisi ollut kehittäjälle erittäin mielenkiintoinen osa-alue syventyä tarkemmin. Valitettavasti aikataulun puitteissa kyseinen julkaisuputki päätettiin siirtää jatkokehityslistalle.

Sopimusprosessin tiedettiin vievän oma aikansa, ja suora yhteys maksurajapinnan tarjoavaan yritykseen saatiin välittömästi. Sopimuksen teko sujui ongelmitta, ja sen hoitaminen ulkoistettiin suoraan tilitystä hoitavalle taholle. Sopimuksen voimaantuloa jouduttiin kuitenkin lopulta odottamaan oman aikansa. Hakuprosessin aloitus muun sovelluskehityksen aikana oli erityisen tärkeää, sillä MobilePay ja Pivo maksutapoja ei pystynyt testaamaan testitunnuksilla.

Ensin tutustuttiin tarkemmin kolmannen osapuolen palveluihin, jotka oli suunnitteluvaiheessa katsastettu potentiaalisiksi vaihtoehdoiksi toiminnallisuudeltaan sekä helppokäyttöisyydeltään. Pettymyksiä alkuperäisten valintojen suhteen ei juuri tullut ja suunnitelmaa päästiin toteuttamaan suunnitellulla tavalla.

Tärkeimmäksi osa-alueeksi, joka oli syytä validoida, oli Chatfuel-palvelu. Keskustelukulku on käytännössä tärkein toiminto, jonka on syytä toimia moitteetta. Sen rakentaminen ja ylläpito on oltava teknisesti helppoa ja intuitiivista. Teimme muutaman eri variaation siitä, mitä kaikkea käyttäjälle voitaisiin tarjota. Variaatioita läpikäydessä oli kuitenkin palattava MVP malliin. Monet ideat oli siirrettävä jatkokehityslistalle ja pyrkiä suorittamaan toimintoja ulos nopeaan tahtiin. Helposti tuntui alussa uudet toiminnot kiinnostavan ja vievän huomiota varsinaisen maksusovelluksen toteuttamiselta.

Tässä vaiheessa keskustelukulku oli selvä ja Chatfuelin avulla onnistuttiin myös hyödyntämään sijaintitiedot laitteelta. Tämän jälkeen vaatimusten täyttyminen edellyttää enää tietyt aukioloajat maksamiselle. Päädyimme kovakoodaamaan ajastuksen suoraan Node-palvelimelle, joka ohjaa maksuportaaliin. Tämä oli nopea ja toimiva tapa saada sovellusta eteenpäin. Toiminnan jatkokehitys siirrettiin tulevaisuuteen muiden prioriteetiltaan suurempien asioiden edeltä.

Chatfuel integraatiota varten perustettiin Facebookiin testiorganisaatio, jonka tokenilla voitiin rakentaa suljetulle käyttäjäryhmälle prototyyppiversio botista. Botin versiota päivitettiin lähes päivittäin, kun keskustelukulkua alettiin toteuttamaan. Alustavat versiot botista vaikuttivat lupaavilta välittömästi ja projekti kulki eteenpäin hyvää vauhtia.

Keskustelukulun valmistuttua oli selvää mitä parametreja Messengeriltä välittyy verkkopalvelimelle. Niiden avulla voitiin ohjelmoida maksutietojen alustus palvelimella ja toteuttaa uudelleenohjaus maksuportaaliin vaadituin tiedoin. Maksuportaalista palataan määriteltyihin osoitteisiin sen perusteella onnistuiko maksu vai ei, joten niiden toiminnallisuus oli myös tehtävä ja varauduttava samalla erilaisiin virhetilanteisiin.

Seuraavaksi oli varmistettava miten luotu botti saadaan siirrettyä testiorganisaation sivulta oikean Facebook-ryhmän piiriin. Tähän löytyi onneksi kätevä duplikointiominaisuus suoraan Chatfuel-palvelusta. Toiminnallisuudella onnistuttiin samalla tekemään toinen versio botista, jota voitiin käyttää jatkokehityksessä uusien prototyyppien testaukseen.

Sopimuksen valmistuttua lopulliset toimivat kauppiastunnisteet toimitettiin ja otettiin käyttöön. Tämä sujui täysin ongelmitta ja muutos oli yksinkertaista tehdä valitulla ympäristömuuttujia käyttävällä toteutustavalla, kuten esiteltiin aliluvun 3.3 ohjelmassa 2. Koska MobilePay ja Pivo maksutavat eivät toimineet testitunnuksilla, oli niiden toiminta testattava nyt kattavasti. Yllätykseksemme kaikki toimi ilman ongelmatilanteita ja maksut rekisteröityivät aiemmin testatulla tavalla, eikä maksutapojen käyttöönotto vaatinut yhtään lisätyötä.

Viimeinen vaihe toteutuksen osalta oli varsinaisen sovelluksen julkaisu. Se tehtiin syksyllä 2017 ja vastaanotto oli erinomainen. Julkaisusta ja käyttöönotosta lisää seuraavalla aliluvussa.

4.2 Käyttöönotto

Mobiilimaksusovelluksen ensimmäinen versio otettiin käyttöön syksyllä 2017, jolloin se julkaistiin virallisesti. Tulevaa maksutapaa mainostettiin muutaman kerran ennen varsinaista julkaisua, mutta sovelluksen teknisistä käyttövaatimuksista tai käyttöohjeista ei kirjoitettu tarkemmin. Muutaman ensimmäisen jumalanpalveluksen jälkeen käyttäjämäärä oli noin 50 käyttäjän paikkeilla. Käyttäjämäärä on hiljalleen jatkanut kasvuaan, kuten nähdään kuvassa 11.

Kuva 11. *Vuoden 2018 käyttäjien kokonaismäärä analytiikkatyökalussa.*

Varhaisen vaiheen käyttäjiä saatiin odotettua enemmän, mutta täysin ongelmitta ei selvitty. Suurimmaksi haasteeksi osoittautui se, jos Facebook-sovelluksen käytössä oli aiemmin estänyt sijaintipalveluiden käytön. Messenger ei osannut kysyä sitä uudestaan, vaan luotti ettei käyttäjä halua enää jakaa sijaintiaan sovellukselle. Tämä ongelma tuli vastaan erityisesti iPhone käyttäjien kesken, mutta se onnistuttiin ratkaisemaan muuttamalla sovelluksen asetuksia. Asetus muutosten jälkeen sijaintipalveluiden käyttö onnistui ongelmitta.

Toinen ongelma esiintyi maksukortin aktivointiin liittyen. Jostain syystä Pivo-sovelluksen asetukset yliajoivat muiden Siirto-sovellusten käyttömahdollisuuden. Jos Pivoon oli asetettu maksukortin tiedot ja halusi käyttää jotain muuta Siirto-sovellusta, johon oli myös aktivoitu maksukortti, maksuportaali ohjasi silti aina Pivoon oletuksena. Tämä aiheutti merkittävästi hämmennystä käyttäjien keskuudessa, ja vei aikaa sovelluksen kehittäjältä, että löydettiin mistä kyseinen ongelma johtui.

Chatfueliin asetettu tapahtuman sijainti ja sen vertaus laitteiden sijaintiin, on toiminut alusta alkaen moitteetta yhtä kertaa lukuun ottamatta. Kertaalleen tapahtuman sijainniksi

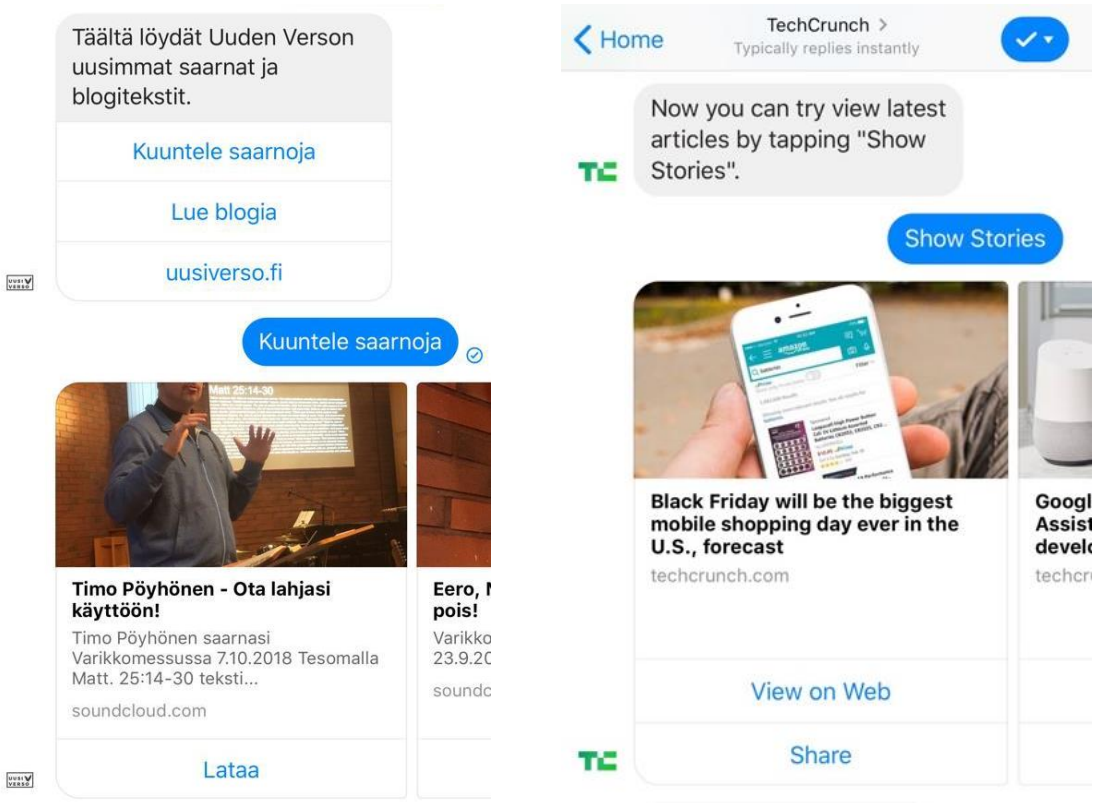
määritelty osoite ei toiminut millään. Vertailu yritettiin toteuttaa aluksi täsmäämällä osoitetta monella eri tapaa. Lopulta yritettiin pelkän postinumeron perusteella vertailla, mutta jostain syystä vertailufunktio palautti aina virheellisen lopputuloksen. Tilaisuus oli kertaluonteinen eikä toistamiseen ole vastaavaa tilannetta tullut ilmi.

4.3 Sovelluksen käyttöliittymän rajoitteet

Sovelluksen käyttöliittymällä tarkoitetaan Facebook Messenger alustalla toteutettua käyttöliittymää eli varsinaista keskustelubotin käyttöliittymää. Käyttöliittymän mahdollisuudet rajautuvat alustan tarjoamiin erilaisiin mallipohjiin (engl. template). Messenger rajapinnan tarjoamat mallipohjat täsmäävät Chatfuelin tarjoamiin komponentteihin. Komponentit toteuttavat sisäisesti tarjotut mallipohjat.

Riippumatta siitä kuka on keskustelubotin keskusteluiden käyttöliittymän toteuttanut, noudattaa se yhtenäistä rajapinnan asettamaa muotoilua jokaisen komponentin osalta. Käytännössä tämä tarkoittaa, että yksittäisen komponentin osalta voidaan muuttaa vain sisältöä. Sisällöksi lukeutuu esimerkiksi Galleria-komponentin tapauksessa yksittäisen sisällön kuva, ja valinnaiset tekstiselitteet. Varsinainen komponentin muotoilu, kulmien pyöristykset, kirjaisinlaji ja muut tyylittelyt määräytyy suoraan rajapinnan mukaan. Näihin ei keskustelubotin kehittäjällä ole mahdollisuutta vaikuttaa.

Kuvassa 12 nähdään kaksi täysin irrallista Messenger alustalle toteutettua keskustelubottia. Vasemmalla puolella tämän työn lopputulos, jossa on listattuna aliluvussa 4.5 esiteltyjä muun toiminnallisuuden sisältöjä. Oikealla puolella nähdään TechCrunch yrityksen jakamaa sisältöä. Molemmissa toteutuksissa alalaidassa nähdään kyseinen Galleria-komponentti, jonka tyylit täsmäävät täysin toisiinsa komponentin varsinaista sisältöä lukuun ottamatta.

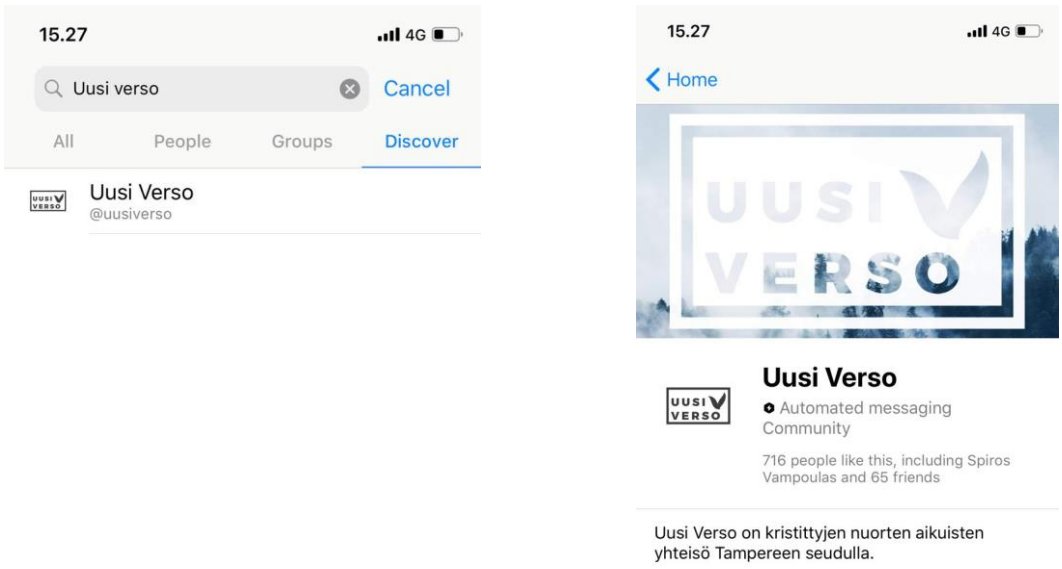


Kuva 12. *Vasemmalla puolella mobiilimaksusovelluksen toteutus Galleria-komponentista, oikealla puolella TechCrunchin toteutus samasta komponentista.*

Tämän työn kannalta näen tyylien rajoitetun yhtenäisyyden vahvuutena kokonaisuutta ajatellen. Yhtenäinen linja säilyy väkisin eri komponenttien välillä, ja ne jakavat saman käyttökokemuksen siitä huolimatta kuka kyseisen keskustelubotin on toteuttanut. Toisaalta myös seuraavassa luvussa 5 käsitelty sovelluksen tuotteistaminen helpottuu kyseisen ominaisuuden takia. Vastuu hienosta ja toimivasta käyttöliittymästä siirretään pois toteuttajalta sekä ylläpitäjältä. Jos muutoksia komponenttien ulkoasuun tulee, ne heijastuvat välittömästi kaikkialle Messenger alustan sisällä. Näin ei myöskään ylläpitäjälle jää vastuuta päivitellä ulkoasua itse.

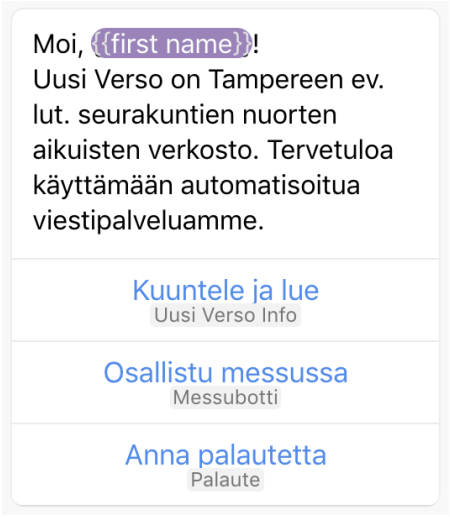
4.4 Keskustelukulku

Keskustelukulun aloitus tapahtuu etsimällä Messengeristä haluttu sivusto tai ryhmä, johon käyttäjällä on oikeus. Valitsemalla hakukoneesta haluttu keskustelukohde, kuvassa 13 vasemmalla puolella, avautuu esille keskustelun ensimmäinen osio, aloitusnäkymä. Aloitusnäkymä on kuvassa 13 oikealla puolella, ja se on visuaalisin osa koko keskustelukulkua.



Kuva 13. *Vasemmalla puolella kuvaa Messenger-sovelluksen hakukone, jossa etsitään sivustoa Uusi Verso. Oikealla klikkauksen jälkeinen aloitusnäkymä.*

Aloitusnäkymän jälkeen varsinainen keskustelu alkaa. Tässä kohtaa keskustelubotti lähettää tervetuloviestin, joka toteutetaan Chatfuel-palvelussa omana lohkonaan. Tervetuloviesti sisältää halutun määrän eri komponentteja. Uuden Verson keskustelubotti lähettää käyttäjälle logonsa, ja etunimellä personoidun tervehdysviestin. Viestin yhteydessä tarjotaan keskustelukulun kaksi eri polkua ryhmiteltyihin toimintoihin sekä painikkeen palautteen lähetystä varten. Tervetuloviestin sisältö nähdään kuvassa 14.



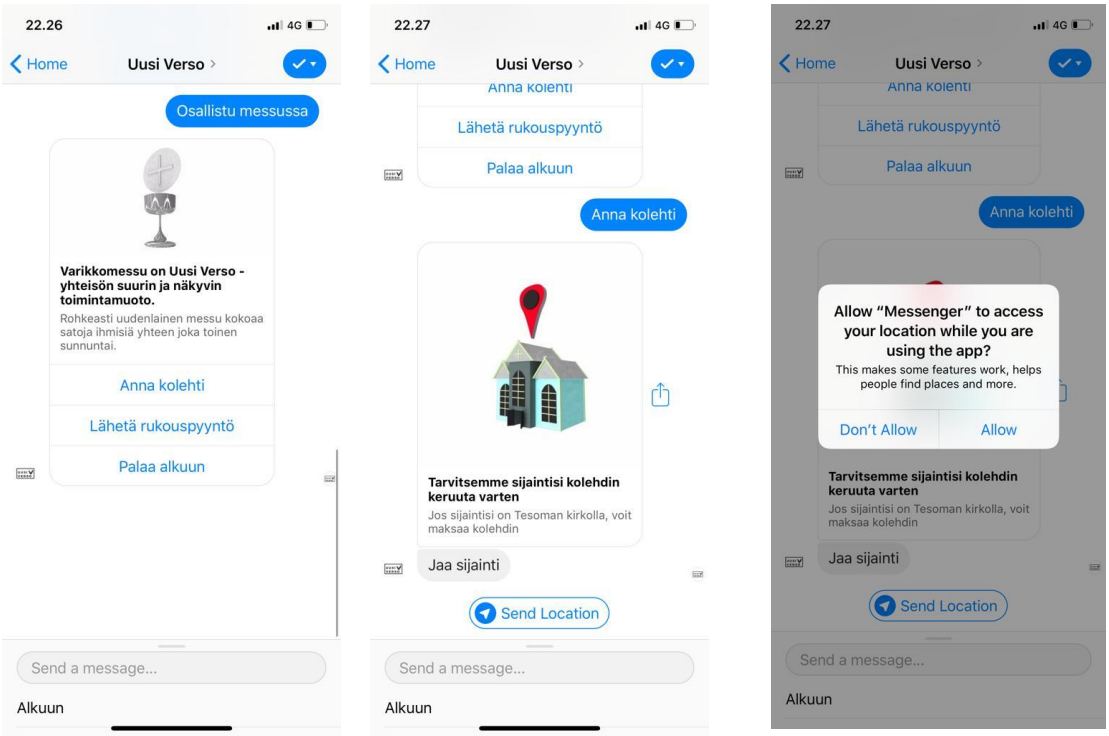
Kuva 14. *Tervetuloviestin tekstisisältö Chatfuel-palvelun keskustelukulun hallintaliittymässä.*

Kuvassa 14 nähdään tapa, jolla saadaan kiinni alustan tarjoamaan rajapintaan käyttäjän tiedoista. Purppuralla maalattu *{{first_name}}* tekstiosuus indikoi, että käytetään alustan tarjoamaa käyttäjäominaisuutta (engl. user attribute) nimeltä *first_name*. Tämä muuttuja

luonnollisesti korvataan varsinaisen viestin lähetyksen yhteydessä käyttäjän asetetulla omalla nimellä. Työn kirjoittajan käyttötapauksessa siinä lukisi ”Moi, Petri!”.

Käyttäjäominaisuuksia hyödyntäen palvelun avulla voidaan luoda käyttäjän syötteestä myös vastaavia itse määriteltyjä muuttujia. Näitä voidaan hyödyntää käyttäjän syötteen hallinnointiin. Tätä ominaisuutta hyödynnetään muutamalla eri tavalla sovelluksen eri vaiheissa. Käyttäjän syöttämä palaute otetaan talteen muuttujaan, jonka arvo ohjataan suoraan sähköpostin lähetys -komponentilla Uuden Verson sähköpostiin. Toinen käyttötapaus ominaisuudelle on sijainnin kysely käyttäjältä. Laitteen sijaintipalvelun välittämä tieto tallennetaan omaan sijaintimuuttujaan, josta saadaan tarvittava tieto jäsennettynä haluttuun muotoon. Sijaintipalvelut välittävät seuraavat tiedot sovellukselle: osoite, postinumero, postitoimipaikka, maa, kunta ja koordinaatit. Näiden avulla voidaan varmuudella täsmätä käyttäjän sijainti tapahtumapaikkaan.

Painamalla painiketta ”Osallistu messussa”, siirrytään kolehdinkeruu-osioon. Tässä osiossa kysytään ja varmistetaan ensin käyttäjän sijainti. Kuvasta 15 nähdään sijainnin määritystä varten eri vaiheet käyttöliittymässä. Ensin navigoidaan toimintoon, ja ensimmäisellä kerralla kysytään sallitaanko alustalle sijaintipalvelujen käyttö sen määrittämistä varten.



Kuva 15. *Käyttöliittymäkuvia sovelluksen sijaintipalveluiden käyttöön liittyen.*

Sijaintitietojen käytön hyväksyntä vaihtelee vähän laitteen käyttöjärjestelmästä riippuen. Yllä olevassa kuvassa nähdään kuvakaappauksia iPhonea käytettäessä, jossa käyttöjärjestelmänä on iOS 12.

Kuvan 16 viimeisen vaiheen jälkeen valitaan jokin tuetuista maksutavoista, Pivo, Siirto tai MobilePay. Muitakin maksutapoja voitaisiin tukea maksuportaalilla, kuten korttimaksua tai tilisiirtoa. Näiden maksutapojen haittapuolena Suomessa on, että maksujen suoritusta varten on tunnistauduttava verkkopankkitunnusten avulla. Painiketta painettaessa Messengeristä avautuu natiivi webview komponentti, joka on käytännössä kevyt selainikkuna sovelluksen sisällä. Selain avataan verkkopalvelimelle ja maksua varten kerätyt tiedot keskustelunaikana välitetään Nodelle HTTP-protokollaa hyödyntäen. Palvelin toteuttaa maksunalustuksen ja uudelleenohjaa valittuun maksutapaan.

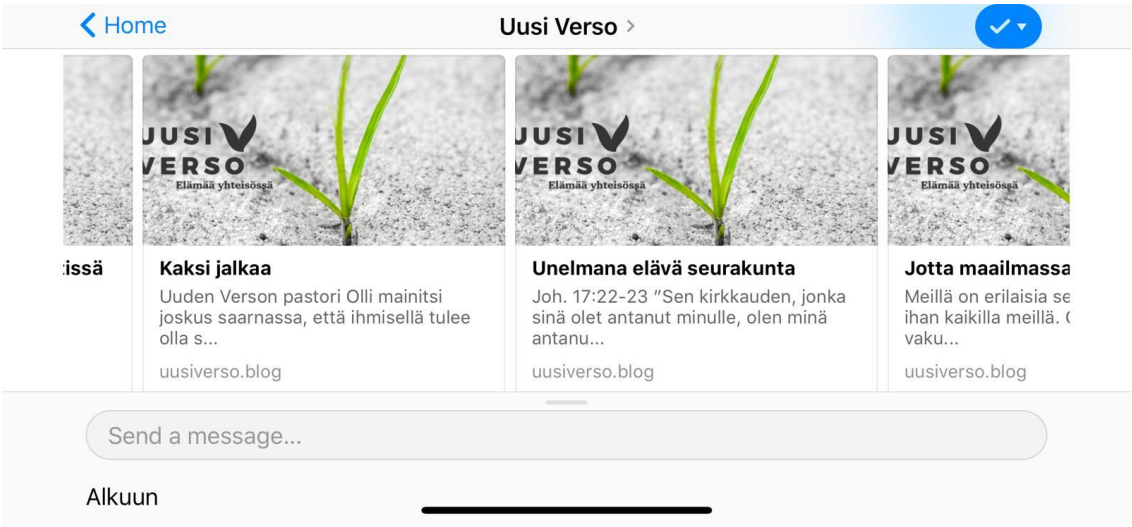
Kuva 16. *Käyttöliittymäkuvia sovelluksessa maksun tekemiseen liittyen.*

Tämän jälkeen maksutavan onnistuessa ohjataan käyttäjä takaisin verkkopalvelimelle, jossa näytetään kiitosviesti onnistuneesta maksusuorituksesta. Epäonnistuessa puolestaan ilmoitetaan virheviestillä epäonnistuneesta maksusta, ja ohjataan käyttäjä yrittämään uudestaan myöhemmin.

Tähän päättyy keskustelukulku kokonaisuudessaan maksun osalta. Seuraavassa aliluvussa käydään läpi mitä sisältöä keskustelukulun toisessa osiossa on tarjolla, kun tervetuloviestin, kuva 14, painiketta ”Kuuntele ja lue” painetaan.

4.5 Muu toiminnallisuus

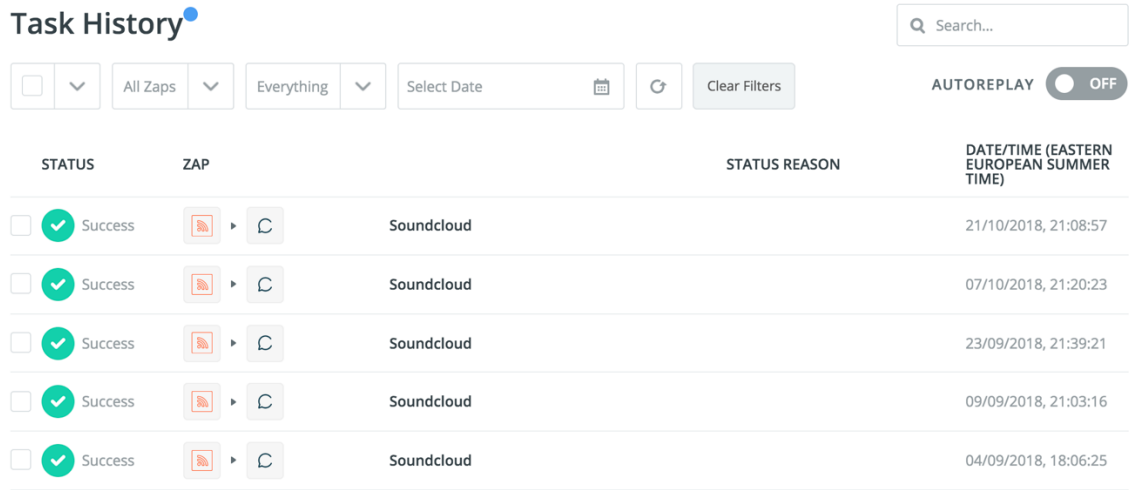
Maksutoiminnallisuuden lisäksi mobiilialusta on erinomainen muun tiedon jakoon saman sovelluksen välityksellä. Keskustelubotin avulla voidaan välittää tärkeää tai kiinnostavaa tietoa käyttäjille liittyen ryhmän toimintaan, kuten nähdään kuvassa 17.



Kuva 17. *Blogitekstejä sovelluksessa.*

Samalla alustalla voidaan siis jakaa tiedotuksia tai uutisia, uusimmat blogikirjoitukset ja vuoroviikoittaiset saarnat ihmisten saataville. Messengerissä tämä käytännössä tarkoittaa eri palveluiden integraatiota Chatfueliin. Sen avulla voidaan hallinnoida eri sisällöt suoraan keskusteluvirtaan.

Muissa sijainnissa sijaitsevien resurssien integraatiot ovat toteutettu muiden toiminnollisuuksien tapaan mahdollisimman ylläpitoystävällisesti hyödyntämällä Chatfuelin kautta Zapier-palvelua, kuvassa 18. Sen avulla voidaan integroida ja automatisoida RSS-syötteiden julkaisu sovellukseen.



Kuva 18. *SoundCloud sisällöt listattuna Zapierissa.*

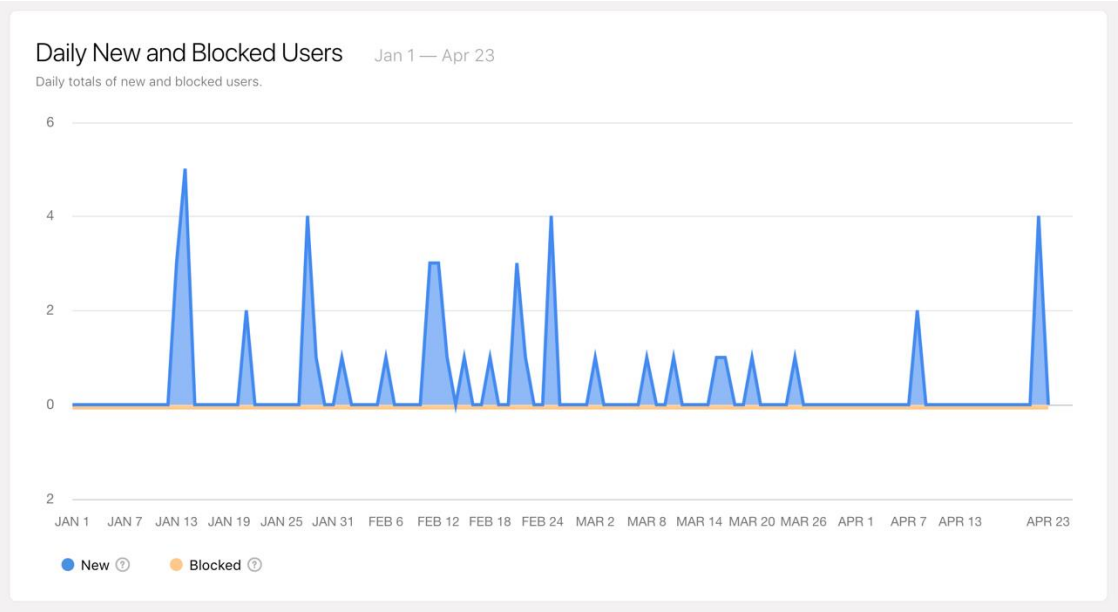
Zapier toimii siten, että yhdelle automatisoinnille asetetaan ensin jokin tapahtuma, ja se laukaisee (engl. trigger) seuraavaksi asetettavan toiminnon (engl. action). SoundCloud ja blogi sisältöjen tapauksessa palvelu kyselee (engl. poll) muutaman minuutin välein RSS-syötteeltä päivityksiä. Mikäli siitä löytyy uutta sisältöä, laukaistaan toiminto. Luodaan

Messengerin keskustelupolkuun uusi komponentti, joka sisältää kuvan, selitteen ja linkin kyseiseen sisältöön, kuten blogikirjoituksien tapauksessa kuvassa 17.

Lisäksi maksusovellusta voidaan kolehtimaksujen lisäksi hyödyntää pienmaksuihin, jotka eivät tarvitse rahankeruulupaa, kuten tapahtumamaksuihin. Keskustelukulkuun voidaan toiseen osioon rakentaa logiikka, jossa ei tarvitse kysyä käyttäjän sijaintia. Tällöin maksu voidaan suoraan suorittaa esimerkiksi tietyllä viitenumerolla. Maksujen vahvistaminen voidaan suorittaa näyttämällä kuitti onnistuneesta maksusuorituksesta, joka on tehty mobiilimaksusovelluksella. On myös mahdollista tarkastaa maksaneiden lista Checkout-maksujärjestelmän käyttöliittymästä, jos maksut ovat tehty tarvittavan ajoissa. Tämä on kätevä tapa suorittaa osallistumismaksuja esimerkiksi leiritoimintaan liittyen, jolloin voidaan varmistaa osallistumisoikeus tapahtumaan.

Messenger-rajapinta tukee myös lähetystoimintaa (engl. broadcasting), jota voidaan hyödyntää jos keskustelun aloittava käyttäjä suostuu tilaamaan (engl. subscribe) keskustelubotin tuottamat sisällöt itselleen. Tässä tapauksessa keskustelubotti voi lähettää esimerkiksi kyseisen viikon kohokohdat tai kaikkien uusien sisältöjen tullessa push-ilmoituksia (engl. push notification).

Chatfuel sisältää myös kattavat analytiikkatyökalut, joiden avulla voidaan seurata käyttäjien toimintaa. Analytiikkaa voidaan hyödyntää monella eri tapaa ja pyrkiä siten esimerkiksi kasvattamaan käyttäjien määrää tai seuraamaan tietyn toiminnallisuuden suosiota verrattuna muihin. Tämän tutkimuksen puitteissa analytiikkaan ei ole kiinnitetty juurikaan huomiota. Kuvaa 19 voidaan kuitenkin esimerkin vuoksi tarkastella liittyen analytiikkatyökalun käyttöön.



Kuva 19. *Kuvakaappaus analytiikkatyökalusta, jossa näkyy päivittäisellä tasolla uniikit uudet käyttäjät.*

Kuvasta 19 nähdään julkaisun jälkeisen kevään 2018 uusien uniikkien käyttäjien määrä päivittäisellä tasolla. Kasvuvauhti ei ole erityisen suuri, mutta uusia käyttäjiä tulee tasaisen jatkuvasti. Uniikkien käyttäjien viikoittainen kasvu on pysynyt suurin piirtein samana kirjoitushetkeen asti.

4.6 Testaus

Elliotin mukaan [17] testit ovat kehittäjän ensimmäinen ja paras keino ohjelmistovirheiden ehkäisyyn. Yleisesti ottaen testeillä voidaan varmistaa ohjelmakoodimuutosten yhteydessä aiempien toiminnollisuuksien toiminta myös muutosten jälkeen. Jos testit ovat riittävän kattavat, ne läpäistäessä voidaan luotettavasti ajaa uusi versio tuotantoon pelkäämättä, että odottamaton virhetilanne ilmaantuu. Kattavat testit mahdollistavat myös ohjelmakoodin kokonaan uudelleen kirjoituksen siten, että voit luotettavasti julkaista uuden version ohjelmasta [18].

Testauksen merkittävyys kasvaa entisestään, kun ohjelmoidaan useamman kehittäjän kesken. Jos testejä ei ole tehty kattavasti, tai jos kehittäjä muokkaa toiminnallisuutta joka aiheuttaa sivuvaikutuksia muualle, on kehittäjän mahdoton tietää hajottaako juuri tehty muutos jonkin toiminnallisuuden muualla ohjelmassa.

Yksikkötestausta käytetään varmistamaan tietyn ohjelmalohkon toimivuus [19], usein esimerkiksi tietyn funktion toimivuuden. Voidaan tarkastella funktiota, määritellä sille parametrit ja verrata funktion tuottamaa lopputulosta odotettuun tulokseen. Yksikkötestejä voidaan pitää siis eräänlaisena bugiraporttina [16].

Erilaisia JavaScriptin testaukseen tarkoitettuja kehyksiä on monia, kuten Jasmine [20], Mocha [21] tai Jest [22]. Usein testauskehykset pyrkivät tarjoamaan kattavan paketin, jolla voi testata lähes mitä ja miten tahansa, samalla monimutkaistamalla testausta merkittävästi. Tape pyrkii ratkaisemaan ongelman eri tavalla [23]. Sillä voidaan luettavasti ja yksinkertaisesti suorittaa yksikkötestit. Voidaan sanoa, että se pakottaa kehittäjän kirjoittamaan yksinkertaisia, selkeää ja modulaarista ohjelmakoodia.

Helpon käyttöönoton ja yksinkertaisuuden takia valittiin juuri Tape, jolla suoritettiin ohjelman yksikkötestaus. Testit voidaan ajaa suoraan komentoriviltä tiedostokohtaisesti tai npm-työkalulla koko testihakemiston testit `npm test` -komennolla. Npm tunnetaan maailman suurimpana JavaScript ohjelmistorekisterinä. Sieltä ladataan kirjastoja käyttöön noin kolme miljardia kertaa viikottain [24]. Ohjelmakirjasto tarjoaa komentorivin käyttöliittymän, myös nimeltään *npm*, jolla paketteja voidaan hallinnoida. Ohjelmassa 5 nähdään esimerkki sovelluksen yksikkötesteistä, jotka on kirjoitettu Tapella.

```
const test = require('tape');
const { upcomingDatesByCount } = require('../index.js');

test('upcomingDatesByCount() should return empty array with count =
0', (t) => {
  const count = 0;
  const result = upcomingDatesByCount(count);
  const expect = [];

  t.deepEqual(result, expect);
  t.end();
});

test('upcomingDatesByCount() should return correct sundays with count
= 1', (t) => {
  const count = 1;
  const result = upcomingDatesByCount(count);
  const expect = new Date('Sun Nov 19 2017 20:30:00 GMT+0200 (EET)');

  t.deepEqual(result, expect);
  t.end();
});

test('upcomingDatesByCount() should return correct sundays with count
= 2', (t) => {
  const count = 2;
  const result = upcomingDatesByCount(count);
  const expect = [new Date('Sun Nov 19 2017 20:30:00 GMT+0200 (EET)'),
new Date('Sun Nov 26 2017 20:30:00 GMT+0200 (EET)')];

  t.deepEqual(result, expect);
  t.end();
});
```

Ohjelma 5. *Tape testauskehyksen avulla kirjoitettuja yksikkötestejä.*

Yllä nähdään logiikka, jolla funktion testaus on toteutettu. Ensin asetetaan parametri, joka annetaan funktiolle, riveillä 6, 16 ja 26. Seuraavaksi määritetään arvo, joka oletetaan palautuvan kyseisen parametrin syötön seurauksesta, riveillä 8, 18 ja 28. Lopulta verrataan täsmääkö odotettu tulos funktion palauttamaan tulokseen, riveillä 10, 20 ja 31. Viimeisenä osana ehkä kaikista tärkein osuus testissä, sen kuvaus. Ilman järkevää kuvausta muut ohjelman kehittäjät eivät välttämättä tiedä mitä ominaisuutta jokin yksikkötesti pyrkii testaamaan. Selkeät yksikkötestit auttavat myös dokumentoimaan ohjelmakoodin rakennetta ja testattavan komponentin käyttötarkoitusta.

Yksikkötestejä on pyritty kirjoittaa testaamaan sovelluksessa käytetyiden funktioiden toteutumista tietyillä parametreilla. Testejä on tehty funktiokohtaisesti useampi, joissa testataan saman funktion palauttamat arvot eri tilanteissa. Rajatapausten testaus on erityisen tärkeää, eikä vain todennäköisten arvojen testaus.

Sovelluksen testejä voitaisiin monipuolistaa ohjelmoimalla esimerkiksi rajapinnan eri tapausten testausta varten tekorajapinta (engl. mock endpoint). Sitä hyödyntäen voitaisiin ajaa testejä ottamatta todellista yhteyttä Checkoutin tarjoamaan rajapintaa. Tätä varten palvelinta täytyisi ajaa testausympäristössä, kuten myös yllä `npm test` -komennolla. Tämä tapa mahdollistaisi testien nopean suorituksen, kun todellisen yhteyden luominen ei olisi tarpeen.

5. JATKOKEHITYSSUUNNITELMA

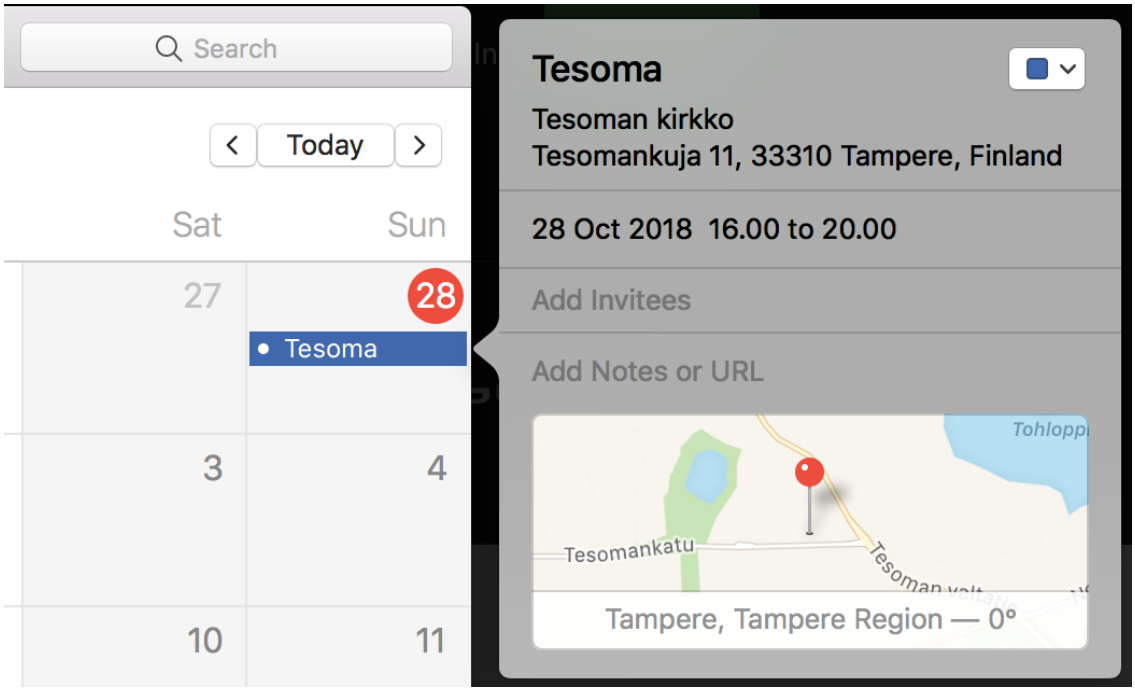
Tällä hetkellä mobiilimaksusovellus toimii hyvin ja todistetusti nykyisessä toimintaympäristössään. Sovelluksen saama merkittävä kiinnostus eri toimijoilta synnytti ajatuksen sovelluksen tuotteistamisesta. Tässä luvussa käymme läpi pohdintoja siitä, mihin suuntaan sovellusta voitaisiin viedä sen tuotteistamiseksi ja miten tiimityöskentely muuttaisi kehitystyön prosessia kokonaisuudessaan. Lopuksi pohdimme muutaman ajatuksen verran sen mahdollista ansaintamallia ja kannattavuutta.

Kiinteästi kovakoodattua lähdekoodissa on tällä hetkellä päivämäärien valinta, jolloin sovellus on käytettävissä. Ajastin asetetaan tietyllä päivämäärälle, josta valitaan listaan joka toinen sunnuntai x-määrä tulevaisuuteen, kuten ohjelmasta 6 nähdään. Listan alkioita verrataan nykyhetkeen ja jos päivämäärä sekä aikahaarukka täsmää, sovelluksen maksutoiminnallisuus otetaan käyttöön.

```
/**
 * Count: Amount of Sundays selected.
 * Returns an array of upcoming Sundays.
 */
function upcomingDatesByCount(count) {
  // GMT time
  const schedule = later.parse.text('at 6:30 pm on Sun');
  const sundays = later.schedule(schedule).next(
    count,
    new Date(2018,1,10)
  );
  return sundays;
}

/**
 * Sundays: Scheduled upcoming Sundays.
 * Returns a reduced array of only every other upcoming Sunday.
 */
function everyOtherSunday(sundays, today) {
  if(sundays.constructor !== Array || sundays.length === 0) return [];
  return sundays.reduce(function(prev, next, index) {
    if (index % 2 === 0 && today.getTime() <= next.getTime()) {
      prev.push(next);
    }
    return prev;
  }, []);
}
```

Ohjelma 6. *Osa sovelluksen aukioloajan määrittämislogiikkaa.*

Ohjelman 6 koodin toteuttama toiminnallisuus olisi syytä korvata kalenterisovellus-integraatiolla, esimerkiksi hyödyntämällä Google kalenterin rajapintaa. Kalenteriin voitaisiin merkitä ajanjaksot, jolloin sovellus pidetään käytettävänä, kuten nähdään esimerkinomaisesti kuvasta 20.



Kuva 20. *Kuvakaappaus Google-kalenterista, johon luotu esimerkkitapahtuma ajan ja paikan hallinnointiin liittyen.*

Yllä olevat tiedot eli päivämäärä aloitus- ja lopetusajalla sekä sijaintitiedot synkronoitaisiin päivittäin mobiilimaksusovellukseen. Tällä tavalla ylläpito päivämäärien suhteen ei olisi teknisesti yhtä haastavaa ja ylläpitovastuu voitaisiin vierittää myös teknisesti vähemmän osaaville käyttäjille tai mahdollisille asiakkaille tuotteistamisen onnistuttua. Google tarjoaa tälle valmiiksi jo oman rajapinnan [25], jota voidaan suoraan hyödyntää toimintoa toteutettaessa.

Muuta tuotteistamiseen liittyvää kehitystyötä olisi Checkout-integraation sisältämän Node.js verkkopalvelun uudelleen ohjelmointi, jossa palvelu siirrettäisiin hyödyntämään Serverless-teknologiaa [26]. Tällöin tuotteen kehitystyö voitaisiin keskittää suoraan tuotekehitykseen ja palvelimista ei tarvitsisi huolehtia. Serverless ympäristön voi luoda suoraan käyttämällä samaa Node.js + Express.js kombinaatiota, jotka ovat jo ennalta tuttuja, ja joilla nykyinen palvelin on toteutettu tällä hetkellä. Serverless teknologiaa hyödyntämällä kehittäjien ei tarvitse huolehtia palvelimen provisoinnista, skaalautuvuudesta, tietokantojen indeksoinnista eikä käyttöjärjestelmätason valinnoista. Teknologialla ajatusmaailma muuttuu virtuaalikoneiden ajosta funktioiden ajoon. [26]

Tuotteistaessa sovellusta mukaan tulisi tiimityöskentely, joka toisi luonnollisesti myös omat haasteensa kehitystyöhön. Jatkuvan integraation ja jatkuvan toimituksen menetelmät olisi tässä vaiheessa mielekästä ottaa käyttöön. Rakennettaisiin julkaisuputki, joka suorittaa kaikki automaattiset testit kaikkia uusia muutoksia vastaan lähdekoodissa. Tällä tavoin vähennettäisiin merkittävästi bugeja, joita tuotantoon väistämättä kulkeutuisi sekä tehtäisiin tiimityöskentelystä huomattavasti luotettavampaa. Jos ohjelmakoodi ajetaan luotettavan ja kattavan testiaineiston läpi, uusien toiminnollisuuksien tai vanhan lähdekoodin parantaminen voidaan toteuttaa nopeilla sykleillä eikä tarvitse pelkää toimintojen hajoamista jossain muualla sovelluksessa.

Tuote voisi elää yhden verkkotunnuksen (engl. domain) alla, jonka aliverkkotunnukset (engl. subdomain) toimisivat asiakkaiden osoitteina. Aliverkkotunnukset yhdistettäisiin tietokannasta tiettyyn kauppiastunnisteeseen ja Facebook ryhmän käyttöoikeustunnisteeseen (engl. token). Rakennettaisiin hallintapaneeli, jolla käyttäjät voivat lisätä oman käyttöoikeustunnisteensa ja rekisteröidä sen käyttöön kyseiselle sivustolle. Tätä toiminnallisuutta varten voisi rakentaa ohjatun asennustoiminnon, joka helppokäyttöisyydellään tekisi Facebook ryhmän ja kauppiastunnisteen integroinnista mahdollisen kenelle tahansa asiakkaalle.

Yksi vaihtoehto olisi myös päästä eroon kokonaan Checkout-maksujärjestelmästä, sillä Facebook tarjoaa alustallaan jo maksuominaisuuden, mutta se toimii vielä tämän tutkimuksen kirjoittamisen aikana pelkästään Yhdysvalloissa. Kenties kyseinen ominaisuus toteutuu myös Suomessa joskus, jolloin Facebookiin yhdistetyllä maksukortilla voitaisiin suoraan suorittaa maksuja. Tämä toiminnallisuus nopeuttaisi maksuprosessia ja parantaisi koko sovelluksen käyttökokemusta, kun voitaisiin luopua kokonaan Messenger-sovelluksesta pois siirtymisestä. Tällaisessa tilanteessa koko maksuprosessi voitaisiin suorittaa Messenger-sovelluksen sisällä.

Facebookin maksuominaisuuden toimiessa samalla poistuisi Node:lla kirjoitettu verkkopalvelu ja koko pilvipalvelun käytön tarve. Verkkopalvelun kehitystyö sekä sen arkkitehtuurin ylläpito, skaalautuvuus ja tuotteistaminen on erittäin suuri osa-alue tuotteistamisen suhteen. Jos maksut voisi suorittaa suoraan Facebookin kautta, koko tuotteistaminen sisältäisi huomattavasti pienemmän työn, kun kauppiastunnisteita sekä muita yksilöiviä tietoja ei tarvitsisi ylläpitää missään tietokannassa.

Viimeisenä isona ohjelmistoprojektina voitaisiin myös rakentaa Chatfuel-palvelun toiminnallisuus omaksi sovellukseksi, joka tarjoaa vastaavasti graafisen käyttöliittymän keskustelupolun rakentamiseen ja siten toteuttaa taustalla varsinaisen Facebook Messenger rajapinnan. Siten mobiilimaksusovellus ei olisi riippuvainen mistään kolmannen osapuolen palvelusta. Vältyttäisiin riskiltä jos jokin palvelu poistuu käytöstä, muuttaa ehtojaan sopimattomaksi tai taloudellisesti kannattamattomaksi.

Ohjelmistotyön lisäksi merkittävää pohdintaa tuotteistamisen kannalta olisi mobiilimaksusovelluksen ansaintamalli. Perittäisiinkö käytöstä jokin kiinteä aloituskonfigurointimaksu jonka jälkeen perittäisiin kuukausimaksua? Vai pelkästään kuukausimaksu joka kattaisi kaiken? Täytyisi miettiä mikä olisi hyvä sopimuksen vähittäispituus ja olisiko mahdollista tarjota palvelu ilmaiseksi käyttöön, mutta periä maksuliikenteestä prosentuaalinen osuus. Nämä kysymykset jäävät toistaiseksi auki ja mikäli tuotteistaminen joskus alkaa, ratkaistaan nämä ja muut kysymykset kun on sen aika.

6. TYÖN ARVIOINTI

Työtä aloittaessa lähes kaikki osa-alueet olivat kehittäjälle täysin uutta osaamisaluetta. Käytännön osuus tehtiin yhden ohjelmistosuunnittelijan voimin, mutta suunnitteluapuna sekä toimintojen hyväksyjänä toimi toinen henkilö, jolla oli vahva tuntemus ja näkemys käyttöympäristöstä. Projektin aivan ensiaskeleissa oli mukana myös teknisen puolen ohjausta aiemmin keskustelubotteja tuottaneelta OP:n työntekijältä.

Vaatimukset projektille saatiin suoraan kirkkolaista ja rahankeruuluvasta, joten lopputulokselle voitiin asettaa selkeä päämäärä. Teknologiavalinnat niiden toteutukselle ja valitut kolmannen osapuolen palvelut osoittautuivat erittäin hyviksi valinnoiksi. Palveluiden käyttöönotto sujui pääsääntöisesti ongelmitta. Kehitystyön aikana saatiin samalla kerättyä paljon jatkokehitystä vaativia toimintoja listalle, joista on selkeä edetä jatkossa eteenpäin.

6.1 Vaatimusten täyttyminen

Asetetut lainsäädännölliset vaatimukset mobiilimaksusovelluksen käyttöön liittyen saatiin täytettyä ja varmennettua aikaisessa vaiheessa projektia. Nämä vaatimukset olivat siis aikaan ja paikkaan sidonnaisuus. Jokainen maksutapahtuma, joka osoitetaan kolehtimaksuksi, täytyy olla yksilöitävissä tiettyyn kolehtiin.

Sijaintitiedot kerättiin Facebookin tarjoaman Messenger-rajapinnan [27] toiminnallisuuden kautta. Käyttäjän laitteen sijaintipalvelulta voidaan lähettää oma sijainti rajapintaan ja tätä arvoa verrataan tapahtuman osoitteeseen.

Aikasidonnaisuus puolestaan ohjelmoitiin Node-palvelimeen kovakoodauksena siten, että maksuportaali on auki vain tilaisuuden aikana. Toteutus valitettavasti vaatii sovelluksen nykytilassa ylläpitotyötä kehittäjältä, mutta jatkokehityssuunnitelmasta löytyy valmis suunnitelma ongelman ratkaisuksi.

6.2 Teknologiapäätökset

Työn toteuttaneelle ohjelmistosuunnittelijalle JavaScript oli kielenä hyvin tuttu monelta vuodelta käyttöliittymäpuolen ohjelmoinnista, mutta Node.js toi mukaan palvelinpuolen ympäristön, joka oli täysin uusi osa-alue. Tämän vuoksi valinta olikin erityisen onnistunut, kun sama kieli on valunut asiakaspäästä palvelimelle. Kokonaisuuden hallinta on kirjoitushetkellä myös erittäin suurta valuuttaa työmarkkinoilla.

DigitalOcean oli onnistunut valinta pilvipalveluntarjoajaksi, sillä palvelulla on aktiivinen yhteisö taustalla, josta sai vastauksen ongelmiin nopeasti. Virtuaalikoneelle Nginxin ja käänteisen välityspalvelimen (engl. reverse proxy) konfigurointiin löytyi myös kattavat

dokumentaatiot ja esimerkit, joita seuraamalla palvelimen asetukset oli luotettavasti toteutettavissa.

Checkout-maksujärjestelmän integraatio sujui ongelmitta avoimen lähdekoodikirjaston [4] ja todella kattavan dokumentaation [5] ansiosta. Tärkeänä ominaisuutena maksujärjestelmän käyttöönoton yhteydessä ilmeni verkkopankkien toimittamat testitunnukset yhdessä testikauppiastunnisteiden kanssa. Näin ollen koko maksuprosessia pystyi testaamaan todella kattavasti eri maksutavoilla heti alusta alkaen. Reilun vuoden aikana syksystä 2017 alkaen, kun maksujärjestelmä otettiin käyttöön, vain yhden kerran se on ollut virhetilassa ja maksuja ei onnistunut tekemään. Vika korjattiin heti seuraavana päivänä, mutta luonnollisesti edellisen päivän kolehti jäi silloin keräämättä mobiilimaksusovelluksella.

Chatfuel-palvelun käyttöönottopäätös oli merkittävin askel koko prosessin aikana. Palvelulla integroitiin suoraan muut sisällöt Zapier-integraatiota hyödyntäen ja varsinaisen keskustelukulun luonti sujui intuitiivisesti jopa vähemmän teknisen ihmisen toimesta. Keskustelukulun ylläpito on mahdollista siirtää siis suoraan pois kehittäjän käsistä. Myös sijaintitietojen kysely ja sen tuloksen prosessointi voitiin suorittaa suoraan Chatfuel-palvelun kautta graafisella käyttöliittymällä. Sisäisesti kyseinen toiminnallisuus käyttää Messenger-alustan rajapintaa, mutta sijaintitietojen tarkastelu ja vertaaminen kolehtipaikkaan on helppoa toteuttaa käyttöliittymän kautta.

Zapier-palvelua käytetiin eri ulkoisten resurssien, kuten blogin ja SoundCloud-sisältöjen integroimiseen Chatfuelin avulla. Valinta oli yksinkertaista tehdä, sillä siihen löytyi valmis integraatiovaihtoehto Chatfuelista. Valitettavasti muuta kilpailevaa vaihtoehtoa ei palvelusta löytynyt, joten päätös jäi ainoan palvelun varaan. Tämä oli harmillista, sillä vanhojen sisältöjen tai pieleen menneen tuonnin (engl. import) uudelleenlataus ilmeni olevan erittäin vaikeaa. Syöte reagoi vain uusien sisältöjen luomiseen, jolloin virhetilanteessa lataamatta jäänyt blogi tai SoundCloud-sisältö täytyy poistaa ja lisätä ulkoiseen resurssiin uudestaan. Kenties tähän olisi voinut löytyä ratkaisu, mutta aikataulusyistä jätettiin tutkimatta sen tarkemmin.

6.3 MVP periaatteen toteutuminen

Toteutustyötä tehdessä pyrittiin noudattamaan MVP (Minimum Viable Product) periaatetta, jossa tavoitteena on saada pienellä vaivalla luotua mahdollisimman tehokkaasti lisäarvoa sovellukselle. Tässä tapauksessa se käytännössä muodostui siihen, että tehdään useita iteraatioita suunnitelluista toiminnoista vaihe vaiheelta aina kun on jotain esiteltävää. Näin saadaan aikaisessa vaiheessa palaute tilaajalta, joka tässä tapauksessa oli seurakuntayhteisön pastori Viljakainen.

Päätös seurata kyseistä metodia oli onnistunut. Palautteen saaminen ajoissa ohjasi toteutustyötä oikeaan suuntaan nopeasti ja turhaa työtä tuli tehtyä odotettua vähemmän. Tilanteita syntyi, jossa suunnitelmat meinasivat paisua liian suuriksi tai korkealentoisiksi, mutta metodin periaatteiden seuraaminen auttoi fokusoitumaan käsissä olevaan tehtävään.

7. YHTEENVETO

Tämän työn päämääränä oli ratkaista kolehdinkeruuta varten vaihtoehtoinen maksutapa, sillä ihmisten mukana kuljettamat käteisvarat vähentyvät tutkitusti hiljalleen ja ihmiset kantavat entistä vähemmän käteistä mukanaan [1]. Sähköistä toimivaa maksutapaa ei ole aiemmin ollut tarjolla, joka toimisi lakisääteisten vaatimusten puitteissa.

Työn lopputuloksena syntyi tiedettävästi valtakunnallisesti ensimmäinen tuotantoon viety mobiilimaksusovellus, joka on viikoittaisessa käytössä Tampereen Uuden Verson toiminnassa. Mobiilimaksusovellus täyttää sille asetetut vaatimukset rahankeruuluvan sekä kirkkolain puitteissa. Nämä vaatimukset ovat aikaan ja paikkaan sidonnaisuus, eli maksutapahtuma on oltava yksilöitävissä tiettyyn tilaisuuteen kolehdinkeruuna. Sijainti määritellään laitteen sijaintipalveluita hyödyntäen ja täsmäämällä kyseinen sijainti tapahtuman osoitteeseen. Näin voidaan varmistaa, että kyseinen henkilö haluaa suorittaa kyseisessä sijainnissa olevaan tapahtumaan maksun. Ajallinen vaatimus puolestaan varmistetaan avaamalla verkkopalvelimen maksuominaisuus vain tapahtumien aikana ennalta määrättyyn aikaan. Näiden lisäksi eri maksupainikkeille voidaan asettaa yksilöivä viitenumero, jos halutaan ohjata rahat eri kohteiden välille.

Sovellus toteutettiin hyödyntämällä useita eri ilmaisia ja helppokäyttöisiä työkaluja, joiden lisäksi otettiin käyttöön suomalainen maksujärjestelmä nimeltä Checkout. Maksujärjestelmä integroitiin verkkopalvelimelle, jota ajetaan käänteisen välityspalvelimen avulla pilvessä DigitalOcean palvelun kautta.

Mobiilimaksusovellus on otettu vastaan positiivisesti yhteisöön ja sen käyttäjämäärät kasvavat analytiikan perusteella hiljalleen, kun ihmiset orientoituvat uuden maksutavan käyttöön. Uusi maksutapa synnytti julkaisun jälkeen melkoisen paljon huomiota myös eri medioiden keskuudessa, kun siitä uutisoitiin usean eri median toimesta. Mielenkiintoa riitti myös erinäisiin haastatteluihin, joita yhteisön pastoria tavoiteltiin ja joiden perusteella kirjoiteltiin julkaisuja [28-32]. Myös muut kiinnostuneet tahot ovat ottaneet yhteyttä saadakseen vastaavan maksutavan käyttöönsä, mutta valitettavasti mobiilimaksusovellus ei vielä ole tuotteena sillä tasolla, että se voitaisiin ottaa käyttöön valtakunnallisesti.

Eri toimijoiden osoittama mielenkiinto herätti vahvasti ajatukset sovelluksen tuotteistamisen suhteen, joita käsiteltiin jatkokehityssuunnitelmia käsittelevässä luvussa 5. Työtä riittää vielä tuotteistamisen suhteen ja on syytä harkita missä määrin ja mihin suuntaan kallisarvoinen aika on käytettävä. Ensisijaisesti tavoitteena on saada tuote sellaiseen kuntoon, että sen viikoittainen ylläpito voidaan vierittää kehittäjältä tuotteen tilaajalle. Näistä tärkein jatkokehitysidea on kalenterisovelluksen integrointi, jolla voitaisiin hallinnoida

maksujärjestelmän käyttöönottoaikoja sekä tapahtuman sijaintia. Näin ollen kalenterisovellukseen voitaisiin suoraan asettaa kokous tai muu tapahtuma indikoimaan kolehdinkeruu hetkeä, eikä niitä tarvitsisi asettaa ohjelmakoodiin logiikkana.

LÄHTEET

[1] O. Herrala, Näin maksaminen muuttuu - maksukortti katoaa, käteinen jää historiaan, web page. Available (accessed 20.8.2018): https://www.kauppalehti.fi/uutiset/nain-maksaminen-muuttuu---maksukortti-katoaa--kateinen-jaa-historiaan/xGRmPdts.

[2] K. Lukka, Konstruktiivinen tutkimusote, web page. Available (accessed 6.11.2018): https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

[3] Chatfuel, Create AI Chat Bot for Facebook, web page. Available (accessed 11.9.2018): https://chatfuel.com/.

[4] T. Kaunisto, JavaScript library for online payments with the Finnish Checkout online bank payment system, web page. Available (accessed 24.10.2018): https://github.com/TuureKaunisto/checkout-api.

[5] Checkout Finland Oy, Checkout API Reference, web page. Available (accessed 25.10.2018): https://checkoutfinland.github.io/legacy-api/.

[6] Jie Ding, Leijie Sha, Xiao Chen, Modeling and evaluating IaaS cloud using performance evaluation process algebra, 2016 22nd Asia-Pacific Conference on Communications (APCC), IEEE, pp. 243-247.

[7] Canonical details BootStack managed OpenStack cloud service, in: v3.co.uk, Incisive Financial Publishing Ltd, 2014.

[8] L. Song, J. Zhang, Load Balancing Method Based on Servers and Reverse Proxy, Journal of Networks, Vol. 8, Iss. 7, 2013, pp. 1609. https://www.openaire.eu/search/publication?articleId=doajarticles::e368b5266b294322ed8bbfed44a893e7.

[9] J. Lengstorf, Deploying a Node.js App to DigitalOcean with SSL, web page. Available (accessed 13.11.2018): https://code.lengstorf.com/deploy-nodejs-ssl-digitalocean/.

[10] R. Corbel, E. Stephan, N. Omnes, HTTP/1.1 pipelining vs HTTP2 in-the-clear: Performance comparison, 2016 13th International Conference on New Technologies for Distributed Systems (NOTERE), pp. 1-6.

[11] S. Tilkov, S. Vinoski, Node.js: Using JavaScript to Build High-Performance Network Programs, IEEE Internet Computing, Vol. 14, Iss. 6, 2010, pp. 80-83.

[12] Advanced, production process manager for Node.js, PM2, web page. Available (accessed 7.11.2018): https://pm2.keymetrics.io/.

[13] Linode, Use NGINX as a Reverse Proxy, Linode, web page. Available (accessed 29.10.2018): https://www.linode.com/docs/web-servers/nginx/use-nginx-reverse-proxy/.

[14] V. Lenarduzzi, D. Taibi, MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product, 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, pp. 112-119.

[15] E. Ries, The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Currency, 2011, 336 p.

[16] M. Soni, End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery, 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), IEEE, pp. 85-89.

[17] E. Elliott, 5 Questions Every Unit Test Must Answer, web page. Available (accessed 26.10.2018): https://medium.com/javascript-scene/what-every-unit-test-needs-f6cd34d9836d.

[18] N. Parsons, Testing Node.js in 2018, web page. Available (accessed 26.10.2018): https://hackernoon.com/testing-node-js-in-2018-10a04dd77391.

[19] J. Rocheleau, JavaScript Unit Testing for Beginners, web page. Available (accessed 26.10.2018): https://designmodo.com/test-javascript-unit/.

[20] Jasmine Documentation, Jasmine, web page. Available (accessed 7.11.2018): https://jasmine.github.io/.

[21] Mocha Documentation, Mocha, web page. Available (accessed 7.11.2018): https://mochajs.org/.

[22] Delightful JavaScript Testing, Jest, web page. Available (accessed 7.11.2018): https://jestjs.io/.

[23] Tap-producing test harness for node and browsers, Tape, web page. Available (accessed 7.11.2018): https://github.com/substack/tape.

[24] About npm, web page. Available (accessed 7.11.2018): https://docs.npmjs.com/getting-started/what-is-npm.

[25] Google, Synchronize Resources Efficiently | Calendar API, Google, web page. Available (accessed 28.10.2018): https://developers.google.com/calendar/v3/sync.

[26] A. Saha, S. Jindal, EMARS: Efficient Management and Allocation of Resources in Serverless, 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), IEEE, pp. 827-830.

[27] Facebook, Quick Replies - Messenger Platform, Facebook, web page. Available (accessed 30.10.2018): https://developers.facebook.com/docs/messenger-platform/send-messages/quick-replies#locations.

[28] S. Sarimaa, Varikkomessussa kolehdin voi nyt lahjoittaa Messengerissä, web page. Available (accessed 13.11.2018): https://www.seurakuntalainen.fi/uutiset/varikkomessussa-kolehdin-voi-nyt-lahjoittaa-messengerissa/.

[29] O. Kokko, Yhä harvemmalla on käteistä mukana – kirkko keksi tavan kerätä kolehtia Facebookin kautta, web page. Available (accessed 13.11.2018): https://www.is.fi/taloussanomat/art-2000005563785.html.

[30] M. Pölkki, Tampereella kolehdin voi maksaa jo mobiilisti – Helsingissä kokeilu ei onnistunut: ”Ihmiset eivät tottuneet”, web page. Available (accessed 13.11.2018): https://www.hs.fi/kotimaa/art-2000005533012.html?share=1fbe85ad4f0d3eeea15d9c5afe63a277.

[31] T. Ellilä, Kirkon uusi tamperelaiskeksintö: kolehtia ei tarvitsekaan enää maksaa käteisellä – "Ajattelimme hypätä suoramaksun yli heti tulevaisuuden maksutapaan", web page. Available (accessed 13.11.2018): https://www.aamulehti.fi/hyvaelama/kirkon-uusi-tamperelaiskeksinto-mobiilikolehti-messengerilla-ajattelimme-hypata-suoramaksun-yli-heti-tulevaisuuden-maksutapaan-200736144.

[32] M. Rämö, Tampereella keksittiin kolehti, jonka voi maksaa Messengerillä – "Seurakunnan on hyvä olla ajan tasalla", web page. Available (accessed 13.11.2018): https://www.tamperelainen.fi/artikkeli/604275-tampereella-keksittiin-kolehti-jonka-voi-maksaa-messengerilla-seurakunnan-on-hyva.